

MacTech®

The Journal of Macintosh Technology and Development

A FIRST
LOOK AT XCODE

Xcode Rising

by Dave Mark



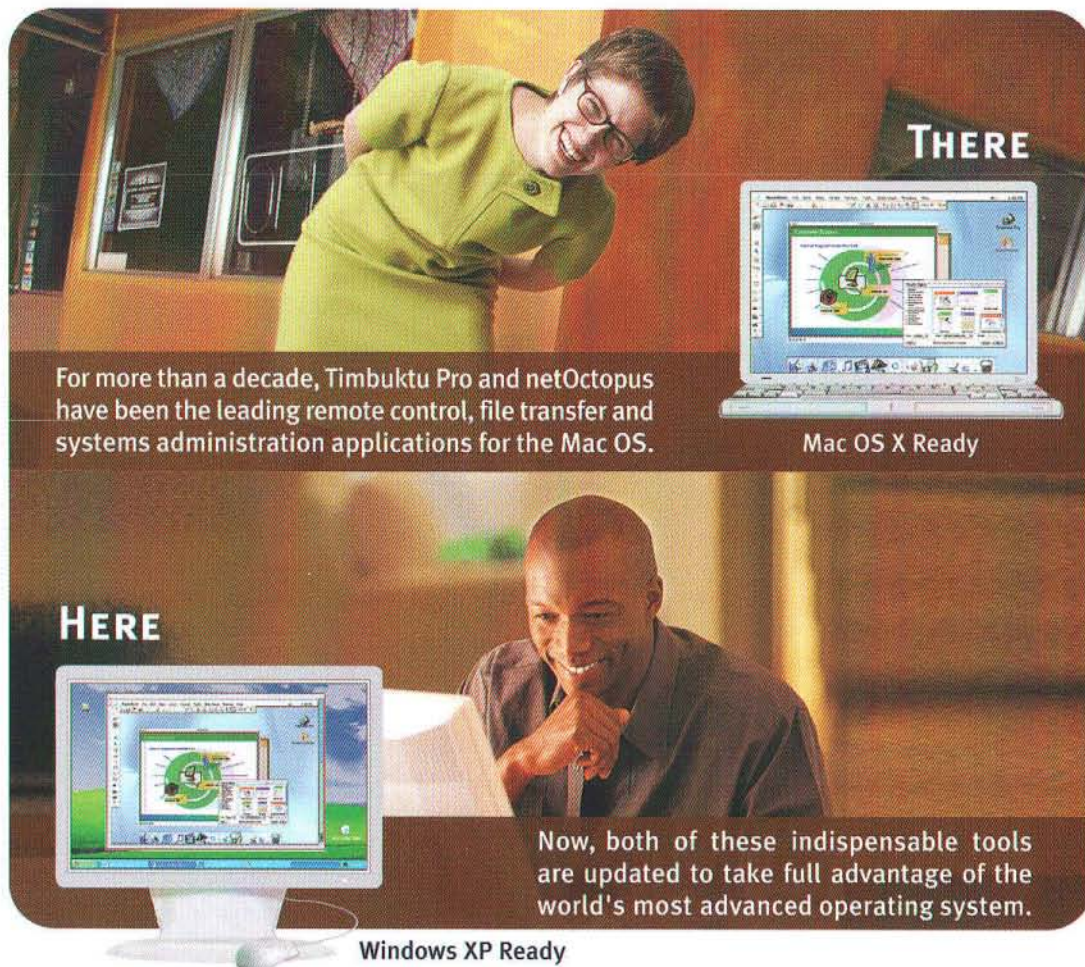
Also in this Issue:

- Using NSParagraphStyle
- Scripting Nisus Writer Express
- Self-Installing Applications
- Movie Review: Revolution OS
- The Process Tree
- Server Side Includes with Apache
- Book Review: Mac OS X Hacks
- Perl Interprocess Communication
- X Files Carbonara
- Apple's Developer Toolchain
- John and Pals' Puzzle Page

\$8.95 US
\$12.95 Canada
ISSN 1067-8360
Printed in U.S.A.



Stay In Control Wherever You Go.



THERE

For more than a decade, Timbuktu Pro and netOctopus have been the leading remote control, file transfer and systems administration applications for the Mac OS.

Mac OS X Ready

HERE

Now, both of these indispensable tools are updated to take full advantage of the world's most advanced operating system.

Windows XP Ready

Timbuktu Pro

Whether you're at home or at work, Timbuktu Pro allows you to operate distant computers as if you were sitting in front of them, transfer files or folders quickly and easily, and communicate by instant message, text chat, or voice intercom.

<http://www.timbuktu.com>

netOctopus

Intuitive and powerful, netOctopus can manage a network of ten or 10,000 computers. Inventory computers, software and devices on your network; distribute software; configure remote computers; and create custom reports on the fly.

<http://www.netoctopus.com>

Learn more, try it, or buy it online. Call us at **1-800-485-5741**.



timbuktu® • netOctopus®

netopia.

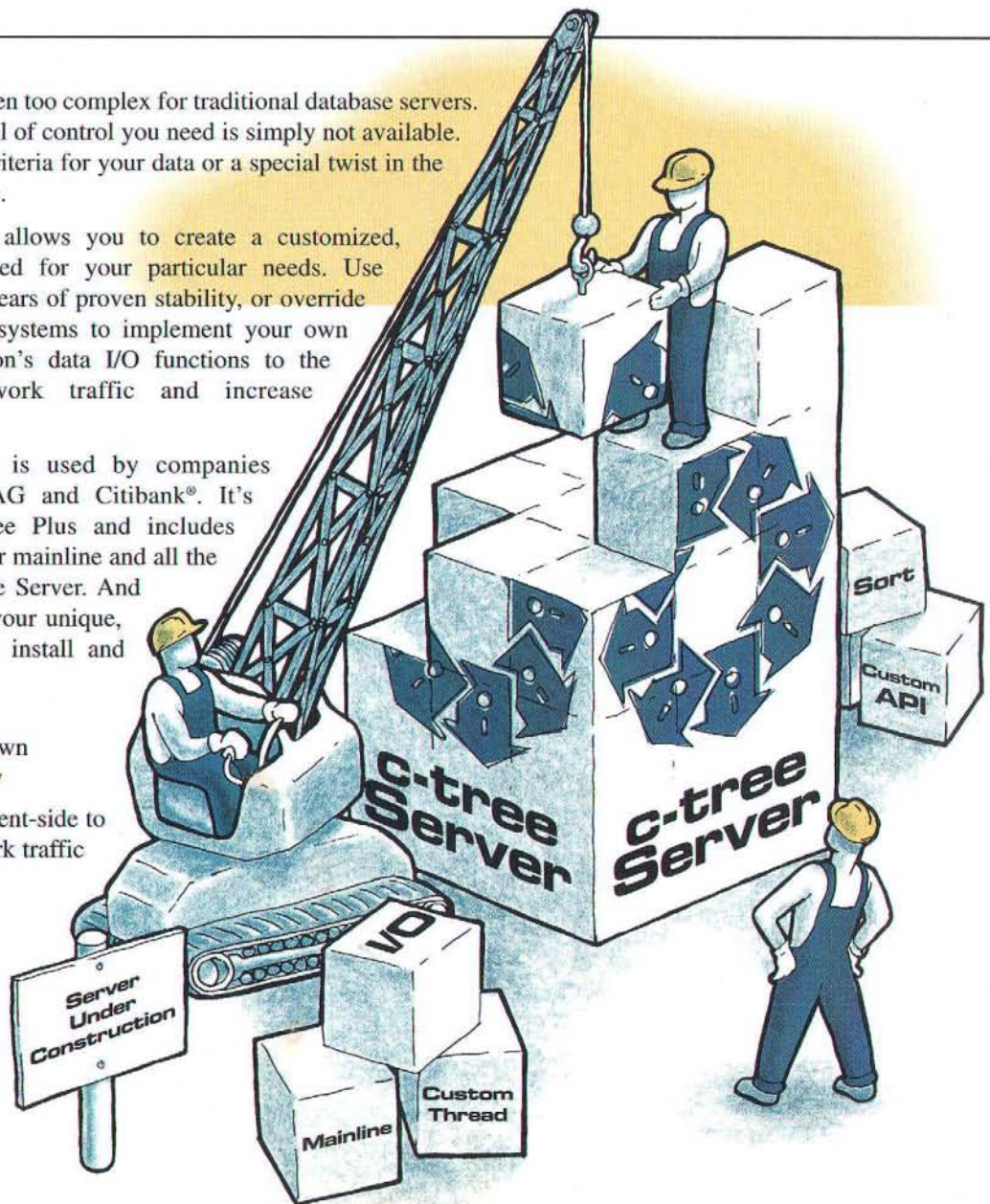
CUSTOMIZE YOUR DATABASE SERVER WITH THE C-TREE® SERVER SDK

Today's database demands are often too complex for traditional database servers. The functionality and precise level of control you need is simply not available. Perhaps you need alternate sort criteria for your data or a special twist in the threading or communication logic.

FairCom's c-tree® Server SDK allows you to create a customized, industrial-strength server designed for your particular needs. Use FairCom's kernel, with over 20 years of proven stability, or override functionality within specific subsystems to implement your own subtleties. Move your application's data I/O functions to the server-side to decrease network traffic and increase performance!

FairCom's c-tree Server SDK is used by companies worldwide such as Software AG and Citibank®. It's integrated seamlessly into c-tree Plus and includes complete source code to the server mainline and all the interface subsystems to the c-tree Server. And best of all, once you've created your unique, customized server, it is easy to install and administer: no DBA required!

- Enhance our server with your own custom server-side functionality
- Move functionality from the client-side to the server-side to reduce network traffic and increase performance
- Modify or replace entire server subsystems
- Complete source for the server mainline, key server subsystems, and client-side
- Flexible OEM licensing



Visit www.faircom.com/ep/mt/sdk today to take control of your server!



FairCom®
www.faircom.com

USA	573.445.6833
EUROPE	+39.035.773.464
JAPAN	+81.59.229.7504
BRAZIL	+55.11.3872.9802

DBMS Since 1979 • 800.234.8180 • info@faircom.com

Other company and product names are registered trademarks or trademarks of their respective owners.

© 2002 FairCom Corporation



Adaptive Server[®]
Enterprise 12.5
MAC OS X

HELPING FILEMAKER PRO CUSTOMERS SCALE TO NEW HEIGHTS.

Want to enhance the performance of FileMaker Pro on MAC OS X? The same database engine that runs Wall Street can help you do just that.

ASE 12.5 increases the reliability, availability and scalability of your FileMaker Pro application. It provides better data integrity, world-class security and the ability to handle thousands of transactions per minute. It'll also give your users the power of SQL queries.

ASE 12.5 for MAC OS X is yet one more example of how

Sybase is helping today's leading companies achieve Information Liquidity: a highly profitable state where all of your information is transformed into real economic value.

A FREE Developer's Edition download is available online at sybase.com/filemaker.

INFORMATION LIQUIDITY.



SYBASE

BETTER WHEN EVERYTHING WORKS TOGETHER.™

How To Communicate With Us

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail**?

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact Developer Depot Customer Service at 877-MACTECH

DEPARTMENTS

Orders, Circulation, & Customer Service

Press Releases

Ad Sales

Editorial

Programmer's Challenge

Online Support

Accounting

Marketing

General

Web Site (articles, info, URLs and more...)

E-Mail/URL

cust_service@mactech.com

press_releases@mactech.com

ad_sales@mactech.com

editorial@mactech.com

prog_challenge@mactech.com

online@mactech.com

accounting@mactech.com

marketing@mactech.com

info@mactech.com

http://www.mactech.com

The MacTech Editorial Staff

Publisher • Neil Ticktin

Editor-in-Chief • Dave Mark

Managing Editor • Jessica Stubblefield

Regular Columnists

Getting Started

by Dave Mark

QuickTime ToolKit

by Tim Monroe

Mac OS X Programming Secrets

by Scott Knaster

Reviews/KoolTools

by Michael R. Harvey

Patch Panel

by John C. Welch

Section 7

by Rich Morin

Untangling the Web

by Kevin Hemenway

John and Pals' Puzzle Page

by John Vink

Regular Contributors

Vicki Brown, Erick Tejkowski,
Paul E. Sevinç

MacTech's Board of Advisors

Jordan Dea-Mattson, Jim Straus
and Jon Wiederspan

MacTech's Contributing Editors

- Michael Brian Bentley
- Vicki Brown
- Marshall Clow
- John. C. Daub
- Tom Djajadiningrat
- Bill Doerrfeld, Blueworld
- Andrew S. Downs
- Richard V. Ford, Packeteer
- Gordon Garb, Sun
- Ilene Hoffman
- Chris Kilbourn, Digital Forest
- Rich Morin
- John O'Fallon, Maxum Development
- Will Porter
- Avi Rappoport, Search Tools Consulting
- Ty Shipman, Kagi
- Chuck Shotton, BIAP Systems
- Cal Simone, Main Event Software
- Steve Sisak, Codewell Corporation
- Chuck Von Rospach, Plaidworks

Xplain Corporation Staff

Chief Executive Officer • Neil Ticktin

President • Andrea J. Sniderman

Controller • Michael Friedman

Production Manager • Jessica Stubblefield

Production/Layout • Darryl Smart

Marketing Manager • Nick DeMello

Account Executive • Lorin Rivers
adsales@mactech.com • 800-5-MACDEV

Events Manager • Susan M. Worley
International • Rose Kemps

Network Administrator • David Breffitt

Accounting • Jan Webber, Marcie Moriarty

Customer Relations • Laura Lane, Susan Pomrantz

Shipping/Receiving • Joel Licardie

Board of Advisors • Steven Geller, Blake Park,
and Alan Carsrud

All contents are Copyright 1984-2003 by Xplain Corporation. All rights reserved. MacTech and Developer Depot are registered trademarks of Xplain Corporation. RadGad, Useful Gifts and Gadgets, Xplain, DevDepot, Depot, The Depot, Depot Store, Video Depot, Movie Depot, Palm Depot, Game Depot, Flashlight Depot, Explain It, MacDev-1, THINK Reference, NetProfessional, NetProLive, JavaTech, WebTech, BeTech, LinuxTech, MacTech Central and the MacTutorMan are trademarks or service marks of Xplain Corporation. Sprocket is a registered trademark of eSprocket Corporation. Other trademarks and copyrights appearing in this printing or software remain the property of their respective holders.

MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 850-P Hampshire Road, Westlake Village, CA 91361-2800. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are \$47.00 per year. Canadian subscriptions are \$59.00 per year. All other international subscriptions are \$97.00 per year. Domestic source code disk subscriptions are \$77 per year. All international disk subscriptions are \$97.00 a year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.

POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

C o n t e n t s

August 2003 • Volume 19, Issue 8

Cocoa Tip

62 Using NSParagraphStyle

How to programmatically set tabs and more in an NSTextView

By Clark Jackson, Tacoma Power

Scripting Within Applications

50 Scripting Nisus Writer Express

Writing perl macros to add functionality

By David Linker

Making Your Applications Self-Installing

70 Self-Installing Applications

Or "How I became Installer Free"

By Kenneth H. Wieschhoff, Jr.

Movie Review

76 Revolution OS

The rise of the free software and open source movements

By Andrew S. Downs

Mac OS X Programming Secrets

Scott Knaster is on vacation in the computerless wilderness. He will return with a fresh column next issue.

Section 7

34 The Process Tree

which processes do what

By Rich Morin

Untangling the Web

44 Server Side Includes with Apache

Including Content within Other Content, And More

By Kevin Hemenway, National Resident

Getting Started

6 Getting Started: Xcode Rising

By Dave Mark, Editor In Chief

Review

78 Book Review: Mac OS X Hacks

By Scott Knaster

Perl

14 Perl Interprocess Communication

By Paul Ammann

Recipes

58 X Files Carbonara

Making Navigation Easier for the Impatient

By Richard Patterson, Los Angeles, CA

Viewpoint

40 Apple's Developer Toolchain

Or, how I learned to appreciate Objective C

By John C. Daub

John and Pals' Puzzle Page

38 Puzzle 2: What's on the menu?

By John A. Vink

Copyright 2003 by Dave Mark

Getting Started: Xcode Rising

Beautiful weather. A gorgeous new venue called Moscone West. And a keynote by Steve Jobs. Ah, the annual pilgrimage to Apple's World Wide Developers Conference. I had a great time and met a lot of you at the conference. Moscone West is across 4th street from Moscone Center, the traditional site of MacWorld Expo. A state of the art facility, WWDC marked its maiden voyage. Yes, we were the first conference ever held here. Cool!

As usual, Steve Jobs did an excellent job with the keynote (check out Scott Knaster's July column for a blow-by-blow account). My favorite part was when Steve rolled out Xcode, Apple's replacement for Project Builder. Everyone who attended the conference got a set of Xcode disks, one install for Panther and another for Jaguar. Since Panther is still not released, I thought I'd focus on the Jaguar version of Xcode.

XCODE IN, PROJECT BUILDER OUT

To start off our guided tour of Xcode, let's take a classic Project Builder project and open it in Xcode. First, launch the Xcode app. You'll find it in `/Developer/Applications`. Once Xcode launches, select *Open* from the *File* menu. Navigate into the Sample Code directory in `/Developer/Examples/AppKit/`. We'll be playing with the *Sketch* project. In Xcode's open dialog, you *could* click on the *Sketch* directory, then scroll through all the files looking for the project file. But do this instead: Click on the *Sketch* directory, then click the *Open* button without selecting a file to open. Xcode will find and open the first project file it encounters in that directory. Cool!

Yeah, I know, why not just find the project file in the Finder and double-click it? Two reasons. First, I wanted to show off Xcode's ability to auto-open the project file. And second, I still have Project Builder installed on my machine. Since Sketch's project is a Project Builder file, when I double-click on the project, the Finder launches Project Builder.

Another solution: In the Finder, click on the project file, then do a Command-I to bring up the Get Info window (see **Figure 1**). Click on the *Open with:* disclosure triangle and select Xcode from the popup menu. If you have not yet run Xcode, select *Other...* from the popup and go find the Xcode app. You can click on the *Change All...* button to make this change for all your Project Builder projects.

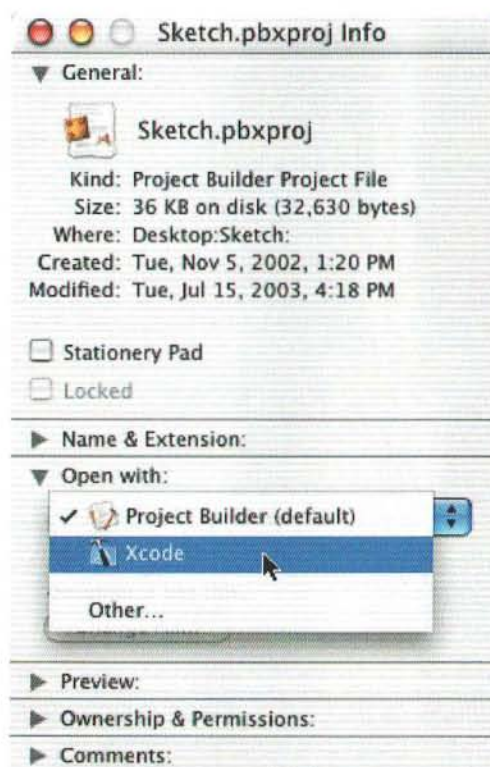


Figure 1. Use the Finder's Get Info window to force your Project Builder projects to open with Xcode.

Dave Mark is a long-time Mac developer and MacTech contributor. Author of more than a dozen books on various Mac-development topics, he's also a founding partner of the nationally syndicated *Online Tonight* and *Net Music Countdown* radio programs. All this and now he's got some fish!

When you click the *Open* button, Xcode will warn you that you are opening an old project file and that it will be upgraded if you continue (See **Figure 2**). Not a prob. Click the *Open* button and let's charge forward.

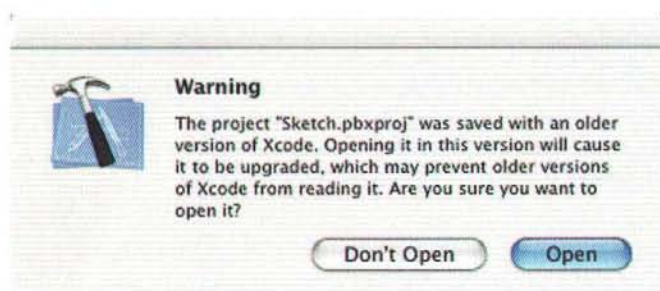


Figure 2. Warning! You are opening an old project file. It will be assimilated.

Figure 3 shows the *Sketch* project window that appears once Xcode upgrades the old Project Builder project file. If your project window looks radically different, perhaps you've mucked with the preferences or customized the toolbar (the strip of icons at the top of the window). To get back the original toolbar, select *Customize Toolbar...* from the *View* menu, drag the default toolbar from the bottom of the window that appears to the toolbar at the top of the window, then click *Done*. Definitely worth spending a minute or two with this, just to get a sense of the kinds of changes you can make to your toolbar.

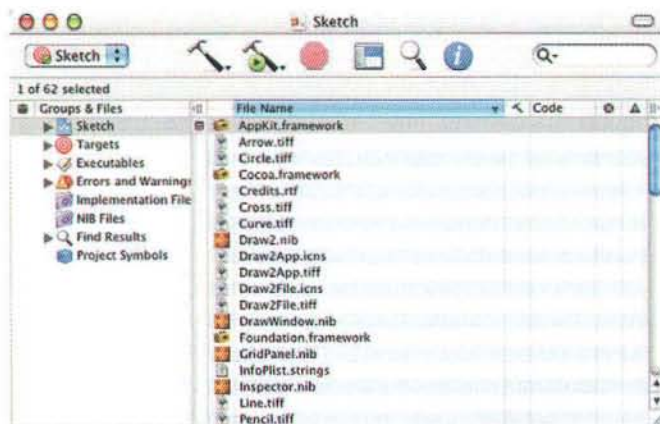


Figure 3. The *Sketch* project window.

A Whole New Build System

When Xcode upgrades an existing project file, it converts the project file format, but does *not* convert the targets themselves. To understand what this means, you have to know a little bit about build systems. Project Builder originally used Make as its build system. Make is a traditional Unix utility that parses dependencies listed in a text-based makefile.



Fetch

Off the leash.

X

Fetchsoftworks.com

Version 4.0.3 now available.

If Make is new to you, take a minute, fire up Terminal, and type in the command `man make`. This will list the man pages for the make command and will give you the basics on build systems.

Over time, Project Builder moved to an open source build system named *Jam*. Here's a link that'll give you the basics on Jam:

<http://www.perforce.com/jam/jam.html>

Jam offers a more compact and sophisticated build language than Make, but is still a text-based build system.

Xcode uses its own, native build system, a build system that is more tightly integrated with the IDE, has a better understanding of the project dependencies and, therefore, produces better build performance. Since the command line and graphical versions of the build system were themselves built from the same frameworks, there is no noticeable performance change when you move between the two.

You can still use the legacy build mechanisms that work with Project Builder. However, you'll need to move to "native builds" if you want to take advantage of Xcode features like Zero Link, Fix and Continue, and Distributed Builds.

Upgrading the Sketch Target

Our next step is to upgrade the *Sketch* target so it uses Xcode's native build system. In the project window's *Groups & Files* pane, click on the *Targets* disclosure triangle to reveal the list of targets in this project. At this point, there is a single target, named *Sketch*. Click on *Sketch*, then select *Upgrade Target to Native* from the *Project* menu.

A panel will appear in the project window that tells you what is about to happen and gives you a chance to name your new target (**Figure 4**). Leave the name as *Sketch (Upgraded)* and click the *Upgrade* button.

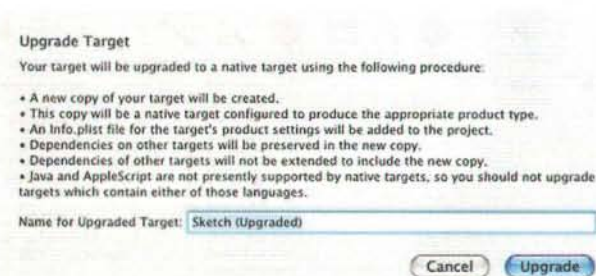


Figure 4. Naming your new, native target.

Now, when you look at the list of targets in the project window, you'll see your new *Sketch (Upgraded)* target. Select *Sketch (Upgraded)* from the popup in the upper-left corner of the project window to make this target the active target

(**Figure 5**). Note that the active target has a small green checkmark in its icon.

Active targets are important! When you do a run, build, or debug, the active target is the one that is run, built, or debugged. You might have one target with full debug info turned on, and a second target that is optimized with no debug info. You might have one target that uses a legacy build system and another that uses Xcode's native build system. Make sure you have the right active target selected before you do your build.

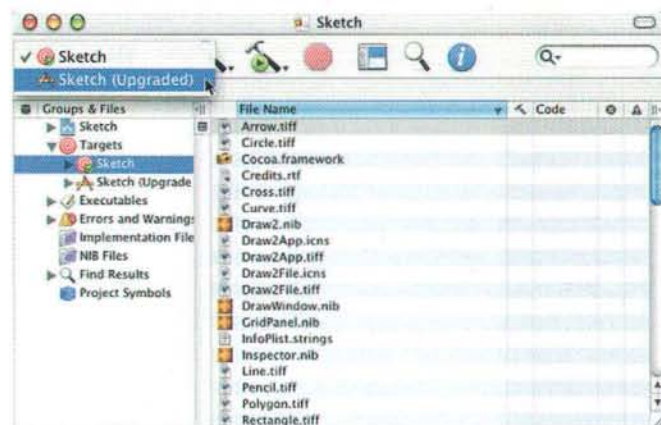


Figure 5. Make the new, *Sketch (Upgraded)* target the active target.

Xcode uses the inspector window to modify target settings. As an example, select the *Sketch (Upgrade)* entry under *Targets*, then bring up the inspector window (either click on the *i-in-a-circle* on the toolbar, or type command-I). In the inspector window, click on the *Build* tab. Each entry in this list is a setting followed by a value. Some values are set via popup menus, some by typing in text, others with checkmarks. Depends on the setting type.

Take a careful look at **Figure 6**. There's a lot going on here. First, I clicked on the *Build* tab, then clicked on the little "drawer" icon in the lower left corner of the inspector window. This opened the drawer to the side, which allows me to select from different sets of settings.

I then clicked on the *Prebinding* setting to select it, then clicked on the split bar at the bottom of the window and dragged it up, revealing an explanation of this setting.

Note also that there is a search field at the bottom of this window. Suppose there were hundreds of settings and I wanted to find anything to do with optimization. If I type "optim" in the search field, the settings are whittled down to two, one of which is called *Optimization level* and the other of which has the word optimization in its explanation text. Very powerful stuff.

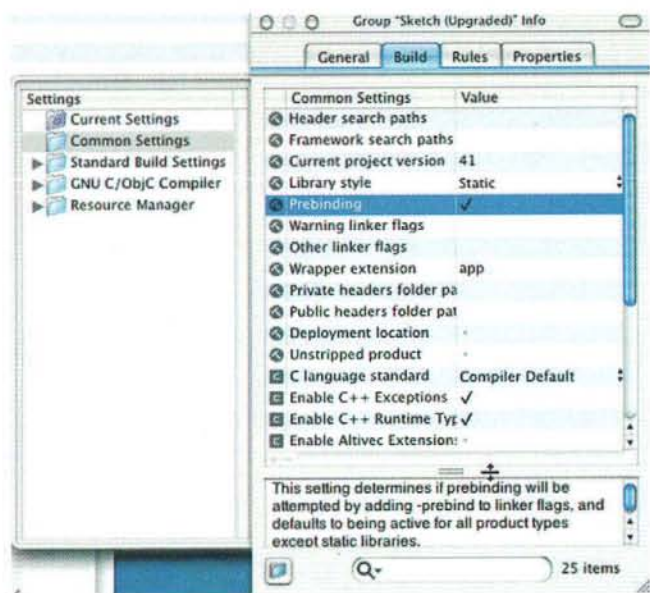


Figure 6. The inspector window, showing the build options for the Sketch (Upgraded) target.

The Groups & Files Pane

On the left side of the project window and just under the active target popup, you'll find the *Groups & Files Pane*. First in this list is the *Files Group*, which lists all the files that make up your project. If you click on this group, you'll see an alphabetical listing of all the project's files, including your Classes, Other Sources (such as main.m and any precompiled headers), Resources (Credits.rtf, Info.plist, InfoPlist.strings, and your nib files), Frameworks, and the Products produced by your project.

Figure 7 shows a default project, based on a Cocoa Document-Based Application template. The five sub-groups that make up the test Files Group are what you get when you create your project. As you might expect, you can create new subgroups (control-click on a group name and select *New Group* from the contextual menu and edit the name of any group (option-click on the name). You can also click and drag any group to move it up and down in the list or place it within another group in the hierarchy. Very intuitive. Very nice.

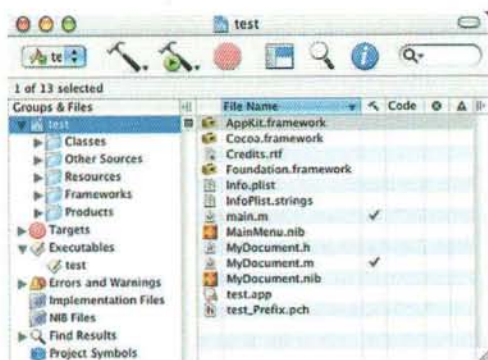
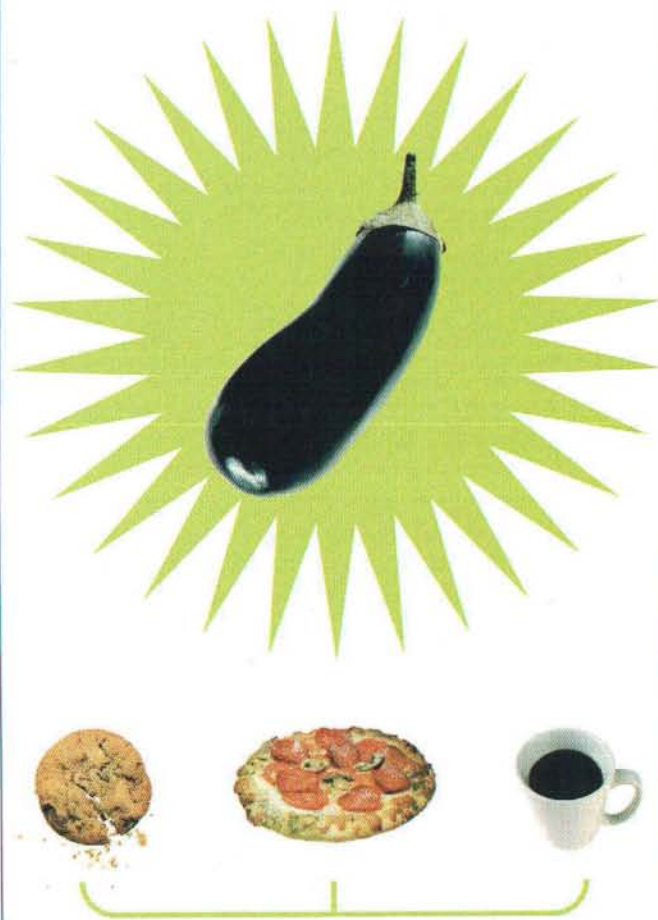


Figure 7. A sample project, showing the default Files Group.

The newest addition
to every **GREAT**
developer's diet.



Easy to learn and use
Complete more testing faster
Test any application on any system
Allows testing of remote systems
Captures test results visually

eggplantTM
Test any application on any system.

The second group in the *Groups & Files* pane is the *Targets* group, which we've already discussed. The third group is your *Executables* group. By default, your project will contain a single executable (see the *test* executable, shown in **Figure 7**). Note that when we upgraded our target in the *Sketch* project, we also created a second executable, called *Sketch (Upgraded)*.

The next group is the *Errors and Warnings* group. Click on this group and the errors and warnings from your last build are shown. Let's give this a try. In the *Sketch* project window, click and hold the first hammer icon in the toolbar, then select *Build* from the popup menu that appears.

Figure 8 shows the result of my *Sketch* build. This build generated 2 warnings, both of them in the file *SKTDrawDocument.m*, and 1 error, in the file *Sketch_main.m* (the warnings are actually part of the shipping *Sketch* project, the error is one I purposefully introduced). If you click on the *Errors and Warnings* group itself, you'll see a list of all the errors and warnings in the project. If you click on one of the file names in the group, you'll see a list of errors and warnings in that file. Finally, if you double-click on the *Errors and Warnings* group, you'll get a separate window with an upper split that lists all the errors and warnings in the project, and a lower split that allows you to edit the source code associated with the selected error or warning. Basically, the project window without the *Groups & Files* pane.

The *Errors & Warnings* group is an example of what Apple calls a *smart group*. Smart groups are dynamic groups that change based on some action or rule. For example, the *Errors & Warnings* group is rebuilt each time you do a build. As you'll see when we get to the *Find Results* group a bit further along, smart groups are a very powerful part of Xcode's arsenal.

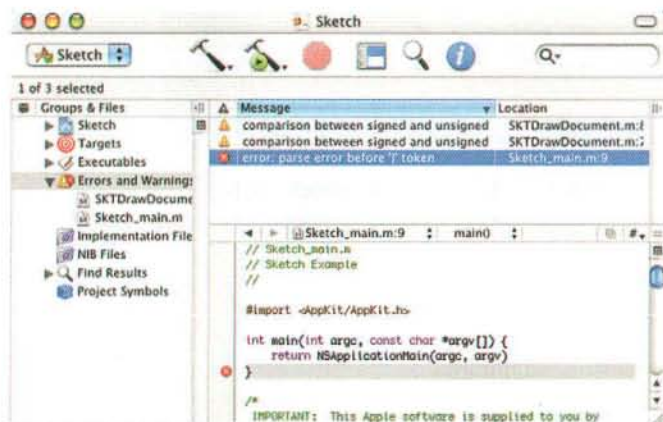


Figure 8. The *Errors and Warnings* group.

Next up is the *Implementation Files* smart group. This is an example of a rule-based smart group. To see what I mean, click on the *Implementation Files* group and then click on the *Show*

info panel icon (an italic i in a blue circle) in the toolbar or type command-i to bring up the inspector window. Rule-based smart groups can be generated using regular expressions or simple patterns. As you can see in the inspector window shown in **Figure 9**, the *Implementation Files* smart group searches the project directory, recursively, to find all files that match the regular expression:

```
?*\.[mcMC]
```

In effect, this regular expression matches any file name that ends in one of ".m", ".c", ".M", or ".C", placing the list of matching file names in the *Implementation Files* smart group. To learn more about regular expressions, check out this link:

<http://www.grymoire.com/Unix/Regular.html>

O'Reilly also has a book on regular expressions (is there nothing they don't cover?) called *Mastering Regular Expressions*, 2nd Edition.

When you click on the *Implementation Files* group, if the main area does not change, bring up the inspector window (command-i), click in the *Using Pattern:* text field, and hit enter. That should force the list to update. A tiny glitch. Probably fixed already!

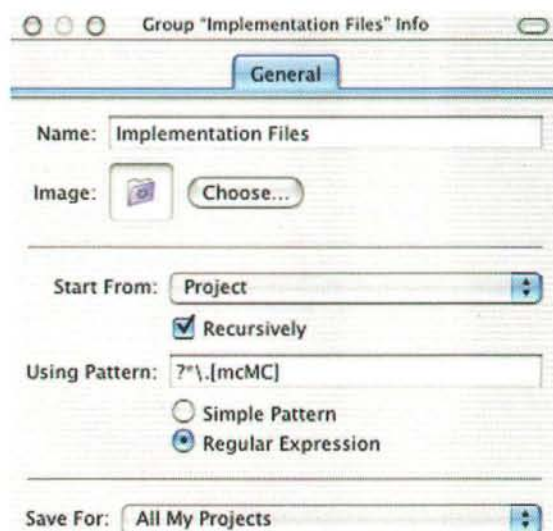


Figure 9. The *Implementation Files* smart group, as seen in the inspector.

UNFAIR



TROLLTECH™

Qt/Mac, the Mac OS X version of the Qt multiplatform C++ application framework, provides a uniquely unfair technological advantage in software development. Qt/Mac lets Mac developers develop in C++ on the platform they love most, while targeting additional operating systems, such as Windows or Linux. Qt/Mac also offers:

- fully object-oriented, elegant API
- single-source, multiplatform development solution
- native compiling and applications that run at native speed

- integration with Project Builder, allowing developers to use Apple's own IDE for editing, compiling and debugging Qt/Mac applications
- OpenGL support, for sophisticated 3D graphics on the Mac
- native look & feel, such as the Aqua style

Don't take our word for it. Visit us at www.trolltech.com/mac. Look at our list of Fortune 500 customers. Read what programmers say about Qt. Download the evaluation version, and see how easy C++ development can be.

Qt /MAC: THE UNFAIR ADVANTAGE IN C++ DEVELOPMENT

The *NIB Files* smart group is just like the *Implementation Files* group, except it uses simple pattern matching instead of regular expression matching, returning all files that match the simple pattern `*.nib`. Smart groups are sweet.

But they get even sweeter. The *Find Results* smart group allows you to refine a search using the search field on the right side of the toolbar. Here's an example. Bring up the *Sketch* project and select *Find in Project...* from the *Find* menu. Enter *draw* in the *Find:* field and check the *Ignore Case* checkbox, then click *Find* (see **Figure 10**).

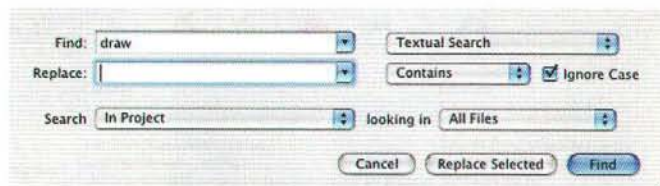


Figure 10. The *Find in Project...* panel.

A list of files should appear in the *detail pane* (the main part of the project window) and the status bar (the thin horizontal strip immediately below the toolbar) should say 1 of 294 selected. So our find just found 294 occurrences of the word "draw" in the project hierarchy. The results of this find were added to the *Find Results* smart group under the name *draw*.

Now let's refine our search a bit. Click on the *draw* subgroup (it may already be selected). In the search field (right

side of the toolbar), type the word "white". You should see 2 matches, for the function *NSDrawWhiteBezel*, both in *SKTGridView.m*. You just did a search within a search. Very cool!

Next on our list is the *Project Symbols* smart group. Click on the *Project Symbols* group and a list of project symbols appears. As you might expect, you can use the search field to filter your symbols. If you click on the magnifying glass icon in the search field, a popup menu appears that lets you select from *Search by All*, *Search by Symbol*, *Search by Symbol Type*, and *Search by Location* (these are the three column headers in the detail area). Pick the type of search you want to do, then start typing. In **Figure 11**, I did a *Search by Symbol Type* and typed in "ma". This reduced the symbol list to show the project macros. Nice!

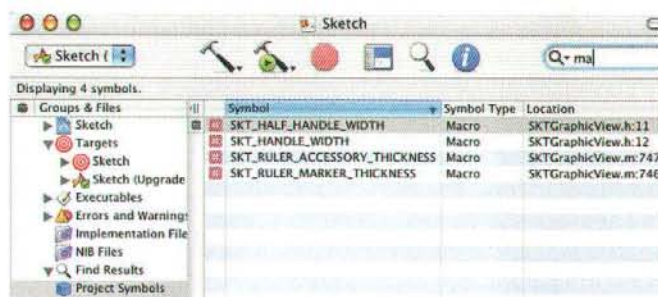


Figure 11. The *Project Symbols* group, with "ma" in the search field, and *Search by Symbol Type* selected in the search field popup menu.

TILL NEXT MONTH...

I've been using Xcode like crazy since WWDC and I find the design very intuitive. Xcode does what I expect it to do. I am impressed, and even more so given that this is prerelease software. Yes, there are a few bugs, but for me they've all been minor. But be smart. Be safe. Backup your projects before you port them to Xcode. After all, prerelease is prerelease.

There are a few interface elements I'd like to see cleaned up. Some are simple refresh issues. Others are behavioral. For example, I'd like to see the little triangle to the right of the magnifying glass icon in the search field only appear when there is a popup menu for that search field's context. And if there is a popup menu, how about including checkmarks next to any selected items so you can go back and see what the current settings are. Don't let these tiny nits deter you even one nanosecond. Xcode is an incredible leap forward.

Before I go, just a few quick things. First, wheel over to <http://developer.apple.com> and check it out. They've completely redone the site. I find it much easier to navigate now, with a lot more detail on the front page. I also wanted to thank (profusely) the folks at Apple who spent so much time answering my Xcode and Cocoa questions. A special shout out to the folks who staffed the OS X and Tools labs at WWDC. You rock!

Whelp, I'm out of space and out of time. Next month, we'll dig a little deeper into Xcode. There is so much more to cover. There's Code Completion, Zero Link, Fix and Continue, and Distributed Builds just to name a few. See you then...☺

stockicons.com

brought to you by the Iconfactory

Icons to go!



Now developers can purchase complete collections of professionally designed icons at an affordable price from the premier source for custom designed icons in the industry. See some of the work we have done for Apple, Microsoft, Aladdin, Intuit, Palm and many others at www.iconfactorydesign.com.

\$349.⁰⁰

Each stock icon collection contains 80 individual icons in three pixel sizes - 32x32, 24x24 and 16x16. Collections are unique in style and are provided in an array of formats including transparent TIFFs, GIF, PNG, .icns, and Windows format .ico's.

For more information visit www.stockicons.com

ching!"
"Cha-Ching!"
"Cha-Ching!"

"Cha-Ching!"

"Cha-Ching!"



New Privilege 6

Generate new software revenue streams — securely, effectively.

Selling, distributing and activating software electronically isn't new. But doing it securely AND all from one source certainly is news! With Privilege from Aladdin, you can have:

Via the Internet:

- A secure ESD process, transforming casual sharing into a super distribution channel for trialware
- Increased customer satisfaction and loyalty with short and simple purchasing process and resumable downloads
- Instant access to international markets with multi-language support

Via digital media and CD:

- Your choice of CD's, or pre-loaded trialware on new computers, with flexible software activation, try-before-you-buy, and try-only
- Capture new opportunities with full control over licensing terms, offers and the user's experience

Take advantage of every new revenue opportunity available today! Call us at 1-800-562-2543, or visit GoPrivilege.com to receive a FREE Privilege information pack.

Privilege[™]
SECURE SOFTWARE eCOMMERCE

By Paul Ammann

Perl Interprocess Communication

This article will discuss the basic IPC facilities of Perl are built out of the good old Unix signals, named pipes, pipe opens, the Berkeley socket routines, and SysV IPC calls. Each is used in slightly different situations.

SIGNALS

Perl uses a simple signal handling model: the %SIG hash contains names or references of user-installed signal handlers. These handlers will be called with an argument which is the name of the signal that triggered it. A signal may be generated intentionally from a particular keyboard sequence like control-C or control-Z, sent to you from another process, or triggered automatically by the kernel when special events transpire, like a child process exiting, your process running out of stack space, or hitting file size limit.

For example, to trap an interrupt signal, set up a handler like this. Do as little as you possibly can in your handler; notice how all we do is set a global variable and then raise an exception. That's because on most systems, libraries are not re-entrant; particularly, memory allocation and I/O routines are not. That means that doing nearly anything in your handler could in theory trigger a memory fault and subsequent core dump.

```
sub catch_zap {
    my $signame = shift;
    $shucks++;
    die "Somebody sent me a SIG$signame";
}
$SIG{INT} = 'catch_zap'; # could fail in modules
$SIG{INT} = \&catch_zap; # best strategy
```

The names of the signals are the ones listed out by **kill -l** on your system, or you can retrieve them from the Config module. Set up an @signame list indexed by number to get the name and a %signo table indexed by name to get the number:

```
use Config;
defined $Config{sig_name} || die "No sigs?";
foreach $name (split(' ', $Config{sig_name})) {
    $signo{$name} = $i;
    $signame[$i] = $name;
    $i++;
}
```

So to check whether signal 17 and SIGALRM were the same, do just this:

```
print "signal #17 = $signame[17]\n";
if ($signo{ALRM}) {
    print "SIGALRM is $signo{ALRM}\n";
}
```

You may also choose to assign the strings 'IGNORE' or 'DEFAULT' as the handler, in which case Perl will try to discard the signal or do the default thing.

On most Unix platforms, the CHLD (sometimes also known as CLD) signal has special behavior with respect to a value of 'IGNORE'. Setting \$SIG{CHLD} to 'IGNORE' on such a platform has the effect of not creating zombie processes when the parent process fails to wait() on its child processes (i.e. child processes are automatically reaped). Calling wait() with \$SIG{CHLD} set to 'IGNORE' usually returns -1 on such platforms.

Some signals can be neither trapped nor ignored, such as the KILL and STOP (but not the TSTP) signals. One strategy for temporarily ignoring signals is to use a local() statement, which will be automatically restored once your block is exited. (Remember that local() values are "inherited" by functions called from within that block.)

```
sub precious {
    local $SIG{INT} = 'IGNORE';
    &more_functions;
}
sub more_functions {
    # interrupts still ignored, for now...
}
```

Sending a signal to a negative process ID means that you send the signal to the entire Unix process-group. This code sends a hang-up signal to all processes in the current process group (and sets \$SIG{HUP} to IGNORE so it doesn't kill itself):

```
{
    local $SIG{HUP} = 'IGNORE';
    kill HUP => -$$;
    # snazzy writing of: kill('HUP', -$$)
}
```

Paul Ammann has been in the IT industry for 16 years, mostly recently working as a Network Security Engineer. He speaks UNIX as a second language. When he is not sitting in front of a computer, he's planning the next vacation adventure with his wife Eve.

Another interesting signal to send is signal number zero. This doesn't actually affect another process, but instead checks whether it's alive or has changed its UID.

```
unless (kill 0 => $kid_pid) {
    warn "something wicked happened to $kid_pid";
}
```

You might also want to employ anonymous functions for simple signal handlers:

```
$$SIG(INT) = sub { die "\nOutta here!\n" };
```

But that will be problematic for the more complicated handlers that need to reinstall themselves. Because Perl's signal mechanism is currently based on the `signal(3)` function from the C library, you may sometimes be so unfortunate as to run on systems where that function is "broken", that is, it behaves in the old unreliable SysV way rather than the newer, more reasonable BSD and POSIX fashion. So you'll see defensive people writing signal handlers like this:

```
sub REAPER {
    $waitedpid = wait;
    # loathe sysV: it makes us not only reinstate
    # the handler, but place it after the wait
    $$SIG(CHLD) = \&REAPER;
}
$$SIG(CHLD) = \&REAPER;
# now do something that forks...
```

or even the more elaborate:

```
use POSIX ":sys_wait_h";
sub REAPER {
    my $child;
    while (($child = waitpid(-1, WNOHANG)) > 0) {
        $kid_status{$child} = $?;
    }
    $$SIG(CHLD) = \&REAPER; # still loathe sysV
}
$$SIG(CHLD) = \&REAPER;
# do something that forks...
```

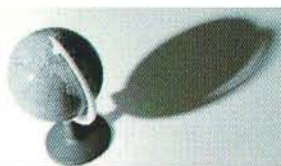
Signal handling is also used for timeouts in Unix. While safely protected within an `eval()` block, you set a signal handler to trap alarm signals and then schedule to have one delivered to you in some number of seconds. Then try your blocking operation, clearing the alarm when it's done but not before you've exited your `eval()` block. If it goes off, you'll use `die()` to jump out of the block, much as you might using `longjmp()` or `throw()` in other languages.

Here's an example:

```
eval {
    local $$SIG(ALRM) = sub { die "alarm clock restart" };
    alarm 10;
    flock(FH, 2); # blocking write lock
    alarm 0;
};
if ($@ and $@ !~ /alarm clock restart/) { die }
```



effigent, Inc.



We deliver efficient and intelligent business solutions

Providing Server and Client enterprise applications for the Mac platform

Services

- Application Development
- Conversions to Mac
- Testing Services
- Maintenance & Support

Solutions

- E - Commerce
- Education Portals
- Knowledge Management
- ERP Implementations

Technology

- Cocoa
- Mac OS X
- WebObjects & J2EE
- Quicktime & Streaming



info@effigent.com

www.effigent.com

Contact Us: 760-723-4800

Copyright 2003, effigent, Inc. All rights reserved.

If the operation being timed out is `system()` or `qx()`, this technique is liable to generate zombies. If this matters to you, you'll need to do your own `fork()` and `exec()`, and kill the errant child process.

For more complex signal handling, you might see the standard POSIX module. Lamentably, this is almost entirely undocumented, but the `t/lib/posix.t` file from the Perl source distribution has some examples in it.

Handling the SIGHUP Signal in Daemons

A process that usually starts when the system boots and shuts down when the system is shut down is called a daemon (Disk And Execution MONitor). If a daemon process has a configuration file which is modified after the process has been started, there should be a way to tell that process to re-read its configuration file, without stopping the process. Many daemons provide this mechanism using the SIGHUP signal handler. When you want to tell the daemon to re-read the file you simply send it the SIGHUP signal.

Not all platforms automatically reinstall their (native) signal handlers after a signal delivery. This means that the handler works only the first time the signal is sent. The solution to this problem is to use POSIX signal handlers if available, their behavior is well-defined.

The following example implements a simple daemon, which restarts itself every time the SIGHUP signal is received. The actual code is located in the subroutine `code()`, which simply prints some debug info to show that it works and should be replaced with the real code.

```
#!/usr/bin/perl -w

use POSIX ();
use FindBin ();
use File::Basename ();
use File::Spec::Functions;

$|=1;

# make the daemon cross-platform, so exec always calls the script
# itself with the right path, no matter how the script was invoked.
my $script = File::Basename::basename($0);
my $SELF = catfile $FindBin::Bin, $script;

# POSIX unmask the sigprocmask properly
my $sigset = POSIX::SigSet->new();
my $action = POSIX::SigAction->new('sigHUP_handler',
                                     $sigset,
                                     &POSIX::SA_NODEFER);
POSIX::sigaction(&POSIX::SIGHUP, $action);

sub sigHUP_handler {
    print "got SIGHUP\n";
    exec($SELF, @ARGV) or die "Couldn't restart: $!\n";
}

code();

sub code {
    print "PID: $$\n";
    print "ARGV: @ARGV\n";
    my $c = 0;
    while (++$c) {
        sleep 2;
        print "$c\n";
    }
}
```

```
}
__END__
```

NAMED PIPES

A named pipe (often referred to as a FIFO) is an old Unix IPC mechanism for processes communicating on the same machine. It works just like a regular, connected anonymous pipes, except that the processes rendezvous using a filename and don't have to be related.

To create a named pipe, use the Unix command `mknod(1)` or on some systems, `mkfifo(1)`. These may not be in your normal path.

```
# system return val is backwards, so && not ||
#
$ENV{PATH} .= ":/etc:/usr/etc";
if(system('mknod', $path, 'p')
    && system('mkfifo', $path))
{
    die "mk(nod,fifo) $path failed";
}
```

A fifo is convenient when you want to connect a process to an unrelated one. When you open a fifo, the program will block until there's something on the other end.

For example, let's say you'd like to have your *signature* file be a named pipe that has a Perl program on the other end. Now every time any program (like a mailer, news reader, finger program, etc.) tries to read from that file, the reading program will block and your program will supply the new signature. We'll use the pipe-checking file test `-p` to find out whether anyone (or anything) has accidentally removed our fifo.

```
chdir; # go home
$FIFO = ".signature";
$ENV{PATH} .= ":/etc:/usr/games";

while (1) {
    unless (-p $FIFO) {
        unlink $FIFO;
        system('mknod', $FIFO, 'p')
            && die "can't mknod $FIFO: $!";
    }

    # next line blocks until there's a reader
    open (FIFO, "> $FIFO") || die "can't write $FIFO: $!";
    print FIFO "John Smith (smith@host.org)\n", 'fortune -s';
    close FIFO;
    sleep 2; # to avoid dup signals
}
```

Deferred Signals

In Perls before Perl 5.7.3 by installing Perl code to deal with signals, you were exposing yourself to danger from two things. First, few system library functions are re-entrant. If the signal interrupts while Perl is executing one function (like `malloc(3)` or `printf(3)`), and your signal handler then calls the same function again, you could get unpredictable behavior—often, a core dump. Second, Perl isn't itself re-entrant at the lowest levels. If the signal interrupts Perl while Perl is changing its own internal data structures, similarly unpredictable behaviour may result.

TECHNICAL DIAGRAMS AND REPORTS

PLOTS • GRAPHS • CHARTS • VISUALIZATION
TEXT • TABLES • FORMS



CONTRACTING

SPECIALIZING IN BUSINESS AND TECHNICAL REPORTING
DATA FROM ANY SOURCE

WHY SETTLE FOR LESS WHEN YOU CAN HAVE THE TOP-MOST EXPERTS?

PROVIDING BUSINESS, ENGINEERING AND SCIENTIFIC REPORT SYSTEMS SINCE 1989

Vvidget Pro™:

A programming kit to control and report data in visual formats.

Peer Visual™:

A server to distribute dynamic visuals to the web and printers.

Vvidget Fusion™:

A distributed and parallel network data acquisition kit.

Vvidget Signals™:

A real-time multi-channel strip chart and signal display kit.

Vvidget User™:

A graph building tool, palette, and preconfigured graphing server.

Limited Time Offer: \$89 (\$49 for students)(this ad made with Vvidget User)

- Technical Analysis
- Real-Time Trading
- Laboratory Instruments
- Process Control
- Database Interfaces
- Modeling
- Device Drivers
- Data Archival
- Mac® OS X Experts

VVI®

www.vvi.com

info@vvi.com

888-VVI-PLOT

There were two things you could do, knowing this: be paranoid or be pragmatic. The paranoid approach was to do as little as possible in your signal handler. Set an existing integer variable that already has a value, and return. This doesn't help you if you're in a slow system call, which will just restart. That means you have to die to `longjmp(3)` out of the handler. Even this is a little cavalier for the true paranoiac, who avoids die in a handler because the system *is* out to get you. The pragmatic approach was to say "I know the risks, but prefer the convenience", and to do anything you wanted in your signal handler, and be prepared to clean up core dumps now and again.

In Perl 5.7.3 and later to avoid these problems signals are "deferred" — that is when the signal is delivered to the process by the system (to the C code that implements Perl) a flag is set, and the handler returns immediately. Then at strategic "safe" points in the Perl interpreter (e.g. when it is about to execute a new opcode) the flags are checked and the Perl level handler from %SIG is executed. The "deferred" scheme allows much more flexibility in the coding of signal handler as we know Perl interpreter is in a safe state, and that we are not in a system library function when the handler is called. However the implementation does differ from previous Perls in the following ways:

Long running opcodes

As Perl interpreter only looks at the signal flags when it about to execute a new opcode if a signal arrives during a long running opcode (e.g. a regular expression operation on a very large string) then signal will not be seen until operation completes.

Interrupting IO

When a signal is delivered (e.g. INT control-C) the operating system breaks into IO operations like read (used to implement Perl's <> operator). On older Perls the handler was called immediately (and as read is not "unsafe" this worked well). With the "deferred" scheme the handler is not called immediately, and if Perl is using system's stdio library that library may re-start the read without returning to Perl and giving it a chance to call the %SIG handler. If this happens on your system the solution is to use :perlio layer to do IO - at least on those handles which you want to be able to break into with signals. (The :perlio layer checks the signal flags and calls %SIG handlers before resuming IO operation.)

Note that the default in Perl 5.7.3 and later is to automatically use the :perlio layer.

Signals as "faults"

Certain signals e.g. SEGV, ILL, BUS are generated as a result of virtual memory or other "faults". These are normally fatal and there is little a Perl-level handler can do with them. (In particular the old signal scheme was particularly unsafe in such cases.) However if a %SIG handler is set the new scheme simply sets a flag and returns as described above. This may cause the operating system to try the offending machine

instruction again and - as nothing has changed - it will generate the signal again. The result of this is a rather odd "loop". In future Perl's signal mechanism may be changed to avoid this - perhaps by simply disallowing %SIG handlers on signals of that type. Until then the work-round is not to set a %SIG handler on those signals. (Which signals they are is operating system dependant.)

Signals triggered by operating system state

On some operating systems certain signal handlers are supposed to "do something" before returning. One example can be CHLD or CLD which indicates a child process has completed. On some operating systems the signal handler is expected to wait for the completed child process. On such systems the deferred signal scheme will not work for those signals (it does not do the wait). Again the failure will look like a loop as the operating system will re-issue the signal as there are un-waited-for completed child processes.

USING OPEN() FOR IPC

Perl's basic open() statement can also be used for unidirectional interprocess communication by either appending or prepending a pipe symbol to the second argument to open(). Here's how to start something up in a child process you intend to write to:

```
open(SPOOLER, "| cat -v | lpr -h 2>/dev/null")
    || die "can't fork: $!";
local $SIG(PIPE) = sub { die "spooler pipe broke" };
print SPOOLER "stuff\n";
close SPOOLER || die "bad spool: $! $?";
```

And here's how to start up a child process you intend to read from:

```
open(STATUS, "netstat -an 2>&1 |")
    || die "can't fork: $!";
while (<STATUS>) {
    next if /^(tcp|udp)/;
    print;
}
close STATUS || die "bad netstat: $! $?";
```

If one can be sure that a particular program is a Perl script that is expecting filenames in @ARGV, the clever programmer can write something like this:

```
% program f1 "cmd1|" - f2 "cmd2|" f3 < tmpfile
```

and irrespective of which shell it's called from, the Perl program will read from the file *f1*, the process *cmd1*, standard input (tmpfile in this case), the *f2* file, the *cmd2* command, and finally the *f3* file. Pretty nifty, eh?

You might notice that you could use backticks for much the same effect as opening a pipe for reading:

```
print grep { !/^(tcp|udp)/ } `netstat -an 2>&1`;
die "bad netstat" if $?;
```

While this is true on the surface, it's much more efficient to process the file one line or record at a time because then you don't have to read the whole thing into memory at once. It also gives you finer control of the whole process, letting you to kill off the child process early if you'd like.

Be careful to check both the `open()` and the `close()` return values. If you're *writing* to a pipe, you should also trap `SIGPIPE`. Otherwise, think of what happens when you start up a pipe to a command that doesn't exist: the `open()` will in all likelihood succeed (it only reflects the `fork()`'s success), but then your output will fail—spectacularly. Perl can't know whether the command worked because your command is actually running in a separate process whose `exec()` might have failed. Therefore, while readers of bogus commands return just a quick end of file, writers to bogus command will trigger a signal they'd better be prepared to handle. Consider:

```
open(FH, "|bogus") or die "can't fork: $!";
print FH "bang\n" or die "can't write: $!";
close FHor die "can't close: $!";
```

That won't blow up until the close, and it will blow up with a `SIGPIPE`. To catch it, you could use this:

```
$SIG{PIPE} = 'IGNORE';
open(FH, "|bogus") or die "can't fork: $!";
print FH "bang\n" or die "can't write: $!";
close FHor die "can't close: status=$?";
```

Filehandles

Both the main process and any child processes it forks share the same `STDIN`, `STDOUT`, and `STDERR` filehandles. If both processes try to access them at once, strange things can happen. You may also want to close or reopen the filehandles for the child. You can get around this by opening your pipe with `open()`, but on some systems this means that the child process cannot outlive the parent.

Background Processes

You can run a command in the background with:

```
system("cmd &");
```

The command's `STDOUT` and `STDERR` (and possibly `STDIN`, depending on your shell) will be the same as the parent's. You won't need to catch `SIGCHLD` because of the double-fork taking place (see below for more details).

Complete Dissociation of Child from Parent

In some cases (starting server processes, for instance) you'll want to completely dissociate the child process from the parent. This is often called daemonization. A well behaved daemon will also `chdir()` to the root directory (so it doesn't prevent unmounting the filesystem containing the directory from which it was launched) and redirect its standard file descriptors from

and to `/dev/null` (so that random output doesn't wind up on the user's terminal).

```
use POSIX 'setsid';

sub daemonize {
    chdir '/' or die "Can't chdir to /: $!";
    open STDIN, '</dev/null' or die "Can't read /dev/null: $!";
    open STDOUT, '>/dev/null' or die "Can't write to /dev/null: $!";
    defined(my $pid = fork) or die "Can't fork: $!";
    exit if $pid;
    setsid or die "Can't start a new session: $!";
    open STDERR, '>&STDOUT' or die "Can't dup stdout: $!";
}
```

The `fork()` has to come before the `setsid()` to ensure that you aren't a process group leader (the `setsid()` will fail if you are). If your system doesn't have the `setsid()` function, open `/dev/tty` and use the `TIOCNOTTY` ioctl() on it instead. See `tty(4)` for details.

Non-Unix users should check their `Your_OS::Process` module for other solutions.

Safe Pipe Opens

Another interesting approach to IPC is making your single program go multiprocess and communicate between (or even amongst) yourselves. The `open()` function will accept a file argument of either `"-|"` or `"|-"` to do a very interesting thing: it forks a child connected to the filehandle you've opened. The

audio recording • editing • effects

Felt Tip
Sound Studio
2.1

Stop

Play

Pause

Record

+12 dB

+6 dB

0 dB

-6 dB

-12 dB

FELT TIP SOFTWARE presents a MAC OS X application "SOUND STUDIO"

available on CD-ROM and as a WEB DOWNLOAD

sales by KAGI written by LUCIUS KWOK

© 2002 Felt Tip Software. All rights reserved.

www.felttip.com/ss

child is running the same program as the parent. This is useful for safely opening a file when running under an assumed UID or GID, for example. If you open a pipe to minus, you can write to the filehandle you opened and your kid will find it in his STDIN. If you open a pipe *from* minus, you can read from the filehandle you opened whatever your kid writes to his STDOUT.

```
use English '-no_match_vars';
my $sleep_count = 0;

do {
    $pid = open(KID_TO_WRITE, "|-");
    unless (defined $pid) {
        warn "cannot fork: $!";
        die "bailing out" if $sleep_count++ > 6;
        sleep 10;
    }
} until defined $pid;

if ($pid) { # parent
    print KID_TO_WRITE @some_data;
    close(KID_TO_WRITE) || warn "kid exited $?";
} else { # child
    ($EUID, $EGID) = ($UID, $GID); # suid progs only
    open (FILE, "> /safe/file")
        || die "can't open /safe/file: $!";
    while (<STDIN) {
        print FILE; # child's STDIN is parent's KID
    }
    exit; # don't forget this
}
```

Another common use for this construct is when you need to execute something without the shell's interference. With `system()`, it's straightforward, but you can't use a pipe open or backticks safely. That's because there's no way to stop the shell from getting its hands on your arguments. Instead, use lower-level control to call `exec()` directly.

Here's a safe backtick or pipe open for read:

```
# add error processing as above
$pid = open(KID_TO_READ, "-|");

if ($pid) { # parent
    while (<KID_TO_READ) {
        # do something interesting
    }
    close(KID_TO_READ) || warn "kid exited $?";
} else { # child
    $EUID, $EGID = ($UID, $GID); # suid only
    exec($program, @options, @args)
        || die "can't exec program: $!";
    # NOTREACHED
}
```

And here's a safe pipe open for writing:

```
# add error processing as above
$pid = open(KID_TO_WRITE, "|-");
$SIG{PIPE} = sub { die "whoops, $program pipe broke" };

if ($pid) { # parent
    for (@data) {
        print KID_TO_WRITE;
    }
    close(KID_TO_WRITE) || warn "kid exited $?";
} else { # child
    ($EUID, $EGID) = ($UID, $GID);
    exec($program, @options, @args)
        || die "can't exec program: $!";
    # NOTREACHED
}
```

Since Perl 5.8.0, you can also use the list form of open for pipes: the syntax

```
open KID_PS, "-|", "ps", "aux" or die $!;
```

forks the `ps(1)` command (without spawning a shell, as there are more than three arguments to `open()`), and reads its standard output via the `KID_PS` filehandle. The corresponding syntax to read from command pipes (with `"|-"` in place of `"-|"`) is also implemented.

Note that these operations are full Unix forks, which means they may not be correctly implemented on alien systems. Additionally, these are not true multithreading. If you'd like to learn more about threading, see the *modules* file mentioned in the SEE ALSO section. *****

Bidirectional Communication with Another Process

While this works reasonably well for unidirectional communication, what about bidirectional communication? The obvious thing you'd like to do doesn't actually work:

```
open(PROG_FOR_READING_AND_WRITING, "| some program |")
```

and if you forget to use the `use warnings` pragma or the `-w` flag, then you'll miss out entirely on the diagnostic message:

```
Can't do bidirectional pipe at -e line 1.
```

If you really want to, you can use the standard `open2()` library function to catch both ends. There's also an `open3()` for tridirectional I/O so you can also catch your child's `STDERR`, but doing so would then require an awkward `select()` loop and wouldn't allow you to use normal Perl input operations.

If you look at its source, you'll see that `open2()` uses low-level primitives like Unix `pipe()` and `exec()` calls to create all the connections. While it might have been slightly more efficient by using `socketpair()`, it would have then been even less portable than it already is. The `open2()` and `open3()` functions are unlikely to work anywhere except on a Unix system or some other one purporting to be POSIX compliant.

Here's an example of using `open2()`:

```
use FileHandle;
use IPC::Open2;
$pid = open2(*Reader, *Writer, "cat -u -n");
print Writer "stuff\n";
$got = <Reader>;
```

The problem with this is that Unix buffering is really going to ruin your day. Even though your `Writer` filehandle is auto-flushed, and the process on the other end will get your data in a timely manner, you can't usually do anything to force it to give it back to you in a similarly quick fashion. In this case, we could, because we gave `cat` a `-u` flag to make it unbuffered. But very few Unix commands are designed to operate over pipes, so this

Hosting Sanctuary for OS X & Xserve

Web Hosting & Development

- MySQL
- Lasso
- Web Objects
- FileMaker Pro

QuickTime Streaming Co-location Services

New Junk Email and Virus Protection from Postini

**Mention this ad and get one month of free hosting
with a 6 month commitment.**

Since 1994 **digital.forest** has provided the most complete Apple Macintosh hosting service in the world. Over the last several years the forest has grown to include service and support for most hosted environments on the internet today. To see how we've grown, please see **www.forest.net** for a summary of our service offerings.

digital.forest



**www.forest.net • info@forest.net
877 720-0483 • 425 483-0483**

seldom works unless you yourself wrote the program on the other end of the double-ended pipe.

A solution to this is the nonstandard *Comm.pl* library. It uses pseudo-ttys to make your program more reasonably:

```
require 'Comm.pl';
$ph = open_proc('cat -n');
for (1..10) {
    print $ph "a line\n";
    print "got back ", scalar <$ph>;
}
```

This way you don't have to have control over the source code of the program you're using. The *Comm* library also has `expect()` and `interact()` functions. Find the library (and I hope its successor *IPC::Chat*) at your nearest CPAN archive.

The newer *Expect.pm* module from CPAN also addresses this kind of thing. This module requires two other modules from CPAN: *IO::Pty* and *IO::Stty*. It sets up a pseudo-terminal to interact with programs that insist on talking to the terminal device driver. If your system is amongst those supported, this may be your best bet.

Bidirectional Communication with Yourself

If you want, you may make low-level `pipe()` and `fork()` to stitch this together by hand. This example only talks to itself, but you could reopen the appropriate handles to `STDIN` and `STDOUT` and call other processes.

```
#!/usr/bin/perl -w
# pipe1 - bidirectional communication using two pipe pairs
# designed for the socketpair-challenged
use IO::Handle; # thousands of lines just for autoflush :-{
pipe(PARENT_RDR, CHILD_WTR); # XXX: failure?
pipe(CHILD_RDR, PARENT_WTR); # XXX: failure?
CHILD_WTR->autoflush(1);
PARENT_WTR->autoflush(1);

if ($pid = fork) {
    close PARENT_RDR; close PARENT_WTR;
    print CHILD_WTR "Parent Pid $$ is sending this\n";
    chomp($line = <CHILD_RDR>);
    print "Parent Pid $$ just read this: '$line'\n";
    close CHILD_RDR; close CHILD_WTR;
    waitpid($pid,0);
} else {
    die "cannot fork: $!" unless defined $pid;
    close CHILD_RDR; close CHILD_WTR;
    chomp($line = <PARENT_RDR>);
    print "Child Pid $$ just read this: '$line'\n";
    print PARENT_WTR "Child Pid $$ is sending this\n";
    close PARENT_RDR; close PARENT_WTR;
    exit;
}
```

But you don't actually have to make two pipe calls. If you have the `socketpair()` system call, it will do this all for you.

```
#!/usr/bin/perl -w
# pipe2 - bidirectional communication using socketpair
# "the best ones always go both ways"

use Socket;
use IO::Handle; # thousands of lines just for autoflush :-{
# We say AF_UNIX because although *.LOCAL is the
# POSIX 1003.1g form of the constant, many machines
# still don't have it.
socketpair(CHILD, PARENT, AF_UNIX, SOCK_STREAM, PF_UNSPEC)
```

```
or die "socketpair: $!";
```

```
CHILD->autoflush(1);
PARENT->autoflush(1);

if ($pid = fork) {
    close PARENT;
    print CHILD "Parent Pid $$ is sending this\n";
    chomp($line = <CHILD>);
    print "Parent Pid $$ just read this: '$line'\n";
    close CHILD;
    waitpid($pid,0);
} else {
    die "cannot fork: $!" unless defined $pid;
    close CHILD;
    chomp($line = <PARENT>);
    print "Child Pid $$ just read this: '$line'\n";
    print PARENT "Child Pid $$ is sending this\n";
    close PARENT;
    exit;
}
```

SOCKETS: CLIENT/SERVER COMMUNICATION

While not limited to Unix-derived operating systems (e.g., WinSock on PCs provides socket support, as do some VMS libraries), you may not have sockets on your system, in which case this section probably isn't going to do you much good. With sockets, you can do both virtual circuits (i.e., TCP streams) and datagrams (i.e., UDP packets). You may be able to do even more depending on your system.

The Perl function calls for dealing with sockets have the same names as the corresponding system calls in C, but their arguments tend to differ for two reasons: first, Perl filehandles work differently than C file descriptors. Second, Perl already knows the length of its strings, so you don't need to pass that information.

One of the major problems with old socket code in Perl was that it used hard-coded values for some of the constants, which severely hurt portability. If you ever see code that does anything like explicitly setting `$AF_INET = 2`, you know you're in for big trouble: An immeasurably superior approach is to use the *Socket* module, which more reliably grants access to various constants and functions you'll need.

If you're not writing a server/client for an existing protocol like NNTP or SMTP, you should give some thought to how your server will know when the client has finished talking, and vice-versa. Most protocols are based on one-line messages and responses (so one party knows the other has finished when a "\n" is received) or multi-line messages and responses that end with a period on an empty line ("\n.\n" terminates a message/response).

Internet Line Terminators

The Internet line terminator is "\015\012". Under ASCII variants of Unix, that could usually be written as "\r\n", but under other systems, "\r\n" might at times be "\015\015\012", "\012\012\015", or something completely different. The standards specify writing "\015\012" to be conformant (be strict in what you provide), but they also recommend accepting a lone "\012" on input (but be lenient in what you require). We haven't always been very good about that in the code in this manpage, but unless you're on a Mac, you'll probably be ok.

Internet TCP Clients and Servers

Use Internet-domain sockets when you want to do client-server communication that might extend to machines outside of your own system.

Here's a sample TCP client using Internet-domain sockets:

```
#!/usr/bin/perl -w
use strict;
use Socket;
my ($remote,$port, $iaddr, $paddr, $proto, $line);

$remote= shift || 'localhost';
$port= shift || 2345;# random port
if ($port =~ /\D/) { $port = getservbyname($port, 'tcp') }
die "No port" unless $port;
$iaddr = inet_aton($remote) || die "no host: $remote";
$paddr = sockaddr_in($port, $iaddr);

$proto = getprotobyname('tcp');
socket(SOCK, PF_INET, SOCK_STREAM, $proto)|| die "socket: $!";
connect(SOCK, $paddr)|| die "connect: $!";
while (defined($line = <SOCK>)) {
    print $line;
}

close (SOCK)|| die "close: $!";
exit;
```

And here's a corresponding server to go along with it. We'll leave the address as INADDR_ANY so that the kernel can choose the appropriate interface on multihomed hosts. If you want to sit on a particular interface (like the external side of a gateway or firewall machine), you should fill this in with your real address instead.

```
#!/usr/bin/perl -Tw
use strict;
BEGIN { $ENV{PATH} = '/usr/ucb:/bin' }
use Socket;
use Carp;
my $EOL = "\015\012";

sub logmsg { print "$0 $$: @_ at ", scalar localtime, "\n" }

my $port = shift || 2345;
my $proto = getprotobyname('tcp');

($port) = $port =~ /^(d+)$/ or die "invalid port";

socket(Server, PF_INET, SOCK_STREAM, $proto)|| die "socket: $!";
setsockopt(Server, SOL_SOCKET, SO_REUSEADDR,
pack("l", 1)) || die "setsockopt: $!";

bind(Server, sockaddr_in($port, INADDR_ANY))|| die "bind: $!";
listen(Server,SOMAXCONN)|| die "listen: $!";

logmsg "server started on port $port";

my $paddr;

$SIG{CHLD} = \&REAPER;

for ( ; $paddr = accept(Client,Server); close Client) {
    my($port,$iaddr) = sockaddr_in($paddr);
    my $name = gethostbyaddr($iaddr,AF_INET);

    logmsg "connection from $name [",
        inet_ntoa($iaddr), "]",
        "at port $port";

    print Client "Hello there, $name, it's now ",
        scalar localtime, $EOL;
}
```

high quality - competitive rates - 16 years experience - award winning

Full Spectrum Software Development & Testing



Device Drivers
Porting Plug-ins
TCP/IP
Carbon / OSX
Cross Platform Development

One Bridge Street
Newton, MA 02458

617-965-0029

www.FullSpectrumSoftware.com

competitive rates - 16 years experience - award winning - high quality

reliable - high quality - competitive rates - efficient - award winning - qa services - reputable

reliable - high quality - competitive rates - efficient - award winning - qa services - reputable

Presto Vivace, Inc.

Fast and Lively Public Relations

Presto Vivace specializes in public relations for small technology companies. Our press contacts database is now available for companies to manage their own publicity.

For only \$99, you can use our professional database to place your press releases. Available in AppleWorks format; e-mail marshall@prestovivace.biz for sample.

4902 Powell Road, Fairfax, VA 22032
703/426-5876, fax 426-5892

<http://www.prestovivace.biz/>

And here's a multihomed version. It's multithreaded in that like most typical servers, it spawns (forks) a slave server to handle the client request so that the master server can quickly go back to service a new client.

```
#!/usr/bin/perl -Tw
use strict;
BEGIN ( $ENV{PATH} = '/usr/ucb:/bin' )
use Socket;
use Carp;
my $EOL = "\015\012";

sub spawn;# forward declaration
sub logmsg ( print "$0 $$: @_ at ", scalar localtime, "\n" )

my $port = shift || 2345;
my $proto = getprotobyname('tcp');

($port) = $port =~ /^(?d+)$/ or die "invalid port";

socket(Server, PF_INET, SOCK_STREAM, $proto) || die "socket: $!";
setsockopt(Server, SOL_SOCKET, SO_REUSEADDR,
    pack("l", 1)) || die "setsockopt: $!";

bind(Server, sockaddr_in($port, INADDR_ANY)) || die "bind: $!";
listen(Server, SOMAXCONN) || die "listen: $!";

logmsg "server started on port $port";

my $waitedpid = 0;
my $paddr;

use POSIX ":sys_wait_h";
sub REAPER {
    my $child;
    while (($waitedpid = waitpid(-1, WNOHANG)) > 0) {
        logmsg "reaped $waitedpid" . ($? ? " with exit $" : "");
    }
    $SIG{CHLD} = \&REAPER;# loathe sysV
}

$SIG{CHLD} = \&REAPER;

for ( $waitedpid = 0;
    ($paddr = accept(Client, Server)) || $waitedpid;
    $waitedpid = 0, close Client)
{
    next if $waitedpid and not $paddr;
    my($port, $iaddr) = sockaddr_in($paddr);
    my $name = gethostbyaddr($iaddr, AF_INET);

    logmsg "connection from $name [",
        inet_ntoa($iaddr), " ]
        at port $port";

    spawn sub {
        $|=1;
        print "Hello there, $name, it's now ", scalar
            localtime, $EOL;
        exec '/usr/games/fortune' #XXX: `wrong' line terminators
            or confess "can't exec fortune: $!";
    };
}

sub spawn {
    my $coderef = shift;

    unless (@_ == 0 && $coderef && ref($coderef) eq 'CODE') {
        confess "usage: spawn CODEREF";
    }

    my $pid;
    if (!defined($pid = fork)) {
        logmsg "cannot fork: $!";
        return;
    } elsif ($pid) {
        logmsg "begat $pid";
        return; # I'm the parent
    }
    # else I'm the child - go spawn
}
```

```
open(STDIN, "<&Client") || die "can't dup client to stdin";
open(STDOUT, ">&Client") || die "can't dup client to
stdout";
## open(STDERR, ">&STDOUT") || die "can't dup stdout to
stderr";
exit &$coderef();
}
```

This server takes the trouble to clone off a child version via fork() for each incoming request. That way it can handle many requests at once, which you might not always want. Even if you don't fork(), the listen() will allow that many pending connections. Forking servers have to be particularly careful about cleaning up their dead children (called "zombies" in Unix parlance), because otherwise you'll quickly fill up your process table.

I suggest that you use the -T flag to use taint checking even if we aren't running setuid or setgid. This is always a good idea for servers and other programs run on behalf of someone else (like CGI scripts), because it lessens the chances that people from the outside will be able to compromise your system.

Let's look at another TCP client. This one connects to the TCP "time" service on a number of different machines and shows how far their clock differ from the system on which it's being run:

```
#!/usr/bin/perl -w
use strict;
use Socket;

my $SECS_of_70_YEARS = 2208988800;
sub ctime { scalar localtime(shift) }

my $iaddr = gethostbyname('localhost');
my $proto = getprotobyname('tcp');
my $port = getservbyname('time', 'tcp');
my $paddr = sockaddr_in(0, $iaddr);
my($host);

$| = 1;
printf "%-24s %8s %s\n", "localhost", 0, ctime(time());

foreach $host (@ARGV) {
    printf "%-24s ", $host;
    my $hiaddr = inet_aton($host) || die "unknown host";
    my $hipaddr = sockaddr_in($port, $hiaddr);
    socket(SOCKET, PF_INET, SOCK_STREAM, $proto) || die
        "socket: $!";
    connect(SOCKET, $hipaddr) || die "bind: $!";
    my $rtime = "";
    read(SOCKET, $rtime, 4);
    close(SOCKET);
    my $hstime = unpack("N", $rtime) - $SECS_of_70_YEARS;
    printf "%8d %s\n", $hstime - time, ctime($hstime);
}
```

Unix-Domain TCP Clients and Servers

That's fine for Internet-domain clients and servers, but what about local communications? While you can use the same setup, sometimes you don't want to. Unix-domain sockets are local to the current host, and are often used internally to implement pipes. Unlike Internet domain sockets, Unix domain sockets can show up in the file system with an ls(1) listing.

```
% ls -l /dev/log
srw-rw-rw- 1 root0 Oct 31 07:23 /dev/log
```

ThinkfreeOffice

COMPATIBLE WITH MICROSOFT WORD, EXCEL AND POWERPOINT

WORDPROCESSOR • SPREADSHEET • PRESENTATION GRAPHICS

The Affordable Office Alternative!



Three High-Performance Applications

Thinkfree Write

Thinkfree Write is a powerful word processing application that enables you to create rich, professional quality documents and Web pages. You can insert tables, images, and clipart, or even apply custom layouts to your document...then effortlessly proofread your work with the easy-to-use spelling and auto-correction features.

Thinkfree Calc

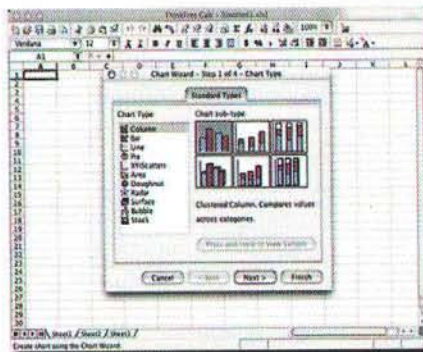
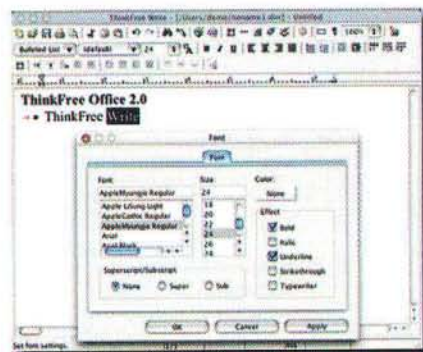
Thinkfree Calc is a full-featured, easy-to-use spreadsheet application that can easily tackle the most complex analytical tasks with over 40 charts and 300 function capabilities. Thinkfree Calc opens, edits, and saves directly into the Microsoft Excel (.xls) format, so users can seamlessly share documents and collaborate with Microsoft Office users.

Thinkfree Show

Thinkfree Show enables you to create high-impact presentations including animation effects, drawings, images, clipart, and other graphic features. Thinkfree Show opens, edits and saves directly into the Microsoft PowerPoint (.ppt) format.

CyberdrivePlus

A free, one-year subscription to CyberdrivePlus is also included. CyberdrivePlus provides you with secure, Internet file storage and free online software upgrades!



"Thinkfree is a best-of-breed program that will exceed your expectations."
— Jeffery Battersby



"Thinkfree Office is the next best thing and then some."
— Deborah Shadovitz



"Thinkfree Office is an impressive attempt to crack the seemingly impenetrable productivity market."
— Chris Ward

amazon.com

Apple Store



Apple Specialist



ONLY
\$49⁹⁵

You can test for these with Perl's -S file test:

```
unless ( -S '/dev/log' ) {
    die "something's wicked with the log system";
}
```

Here's a sample Unix-domain client:

```
#!/usr/bin/perl -w
use Socket;
use strict;
my ($rendezvous, $line);

$rendezvous = shift || '/tmp/catsock';
socket(SOCK, PF_UNIX, SOCK_STREAM, 0) || die "socket: $!";
connect(SOCK, sockadr_un($rendezvous)) || die "connect: $!";
while (defined($line = <SOCK>)) {
    print $line;
}
exit;
```

And here's a corresponding server. You don't have to worry about silly network terminators here because Unix domain sockets are guaranteed to be on the localhost, and thus everything works right.

```
#!/usr/bin/perl -Tw
use strict;
use Socket;
use Carp;

BEGIN { $ENV{PATH} = '/usr/ucb:/bin' }
sub spawn; # forward declaration
sub logmsg { print "$0 $$: @_ at ", scalar localtime, "\n" }

my $NAME = '/tmp/catsock';
my $uaddr = sockadr_un($NAME);
my $proto = getprotobyname('tcp');

socket(Server, PF_UNIX, SOCK_STREAM, 0) || die "socket: $!";
unlink($NAME);
bind (Server, $uaddr) || die "bind: $!";
listen(Server, SOMAXCONN) || die "listen: $!";

logmsg "server started on $NAME";

my $waitedpid;

use POSIX ":sys_wait_h";
sub REAPER {
    my $child;
    while (($waitedpid = waitpid(-1, WNOHANG)) > 0) {
        logmsg "reaped $waitedpid" . ($? ? " with exit $" :
    );
    }
    $SIG{CHLD} = \&REAPER; # loathe sysV
}

$SIG{CHLD} = \&REAPER;

for ( $waitedpid = 0;
    accept(Client, Server) || $waitedpid;
    $waitedpid = 0, close Client)
{
    next if $waitedpid;
    logmsg "connection on $NAME";
    spawn sub {
        print "Hello there, it's now ", scalar localtime,
    "\n";
        exec '/usr/games/fortune' or die "can't exec
    fortune: $!";
    };
}

sub spawn {
    my $coderef = shift;
```

```
unless (@_ == 0 && $coderef && ref($coderef) eq 'CODE')
{
    confess "usage: spawn CODEREF";
}

my $pid;
if (defined($pid = fork)) {
    logmsg "cannot fork: $!";
    return;
} elsif ($pid) {
    logmsg "begat $pid";
    return; # I'm the parent
}
# else I'm the child - go spawn

open(STDIN, "<&Client") || die "can't dup client to
stdin";
open(STDOUT, ">&Client") || die "can't dup client to
stdout";
## open(STDERR, ">&STDOUT") || die "can't dup stdout to
stderr";
exit &$coderef();
}
```

As you see, it's remarkably similar to the Internet domain TCP server, so much so, in fact, that we've omitted several duplicate functions--spawnO, logmsgO, ctimO, and REAPERO--which are exactly the same as in the other server.

So why would you ever want to use a Unix domain socket instead of a simpler named pipe? Because a named pipe doesn't give you sessions. You can't tell one process's data from another's. With socket programming, you get a separate session for each client: that's why acceptO takes two arguments.

For example, let's say that you have a long running database server daemon that you want folks from the World Wide Web to be able to access, but only if they go through a CGI interface. You'd have a small, simple CGI program that does whatever checks and logging you feel like, and then acts as a Unix-domain client and connects to your private server.

TCP CLIENTS WITH IO::Socket

For those preferring a higher-level interface to socket programming, the IO::Socket module provides an object-oriented approach. IO::Socket is included as part of the standard Perl distribution as of the 5.004 release. If you're running an earlier version of Perl, just fetch IO::Socket from CPAN, where you'll also find modules providing easy interfaces to the following systems: DNS, FTP, Ident (RFC 931), NIS and NISPlus, NNTP, Ping, POP3, SMTP, SNMP, SLeay, Telnet, and Time--just to name a few.

A Simple Client

Here's a client that creates a TCP connection to the "daytime" service at port 13 of the host name "localhost" and prints out everything that the server there cares to provide.

```
#!/usr/bin/perl -w
use IO::Socket;
$remote = IO::Socket::INET->new(
    Proto=> "tcp",
    PeerAddr => "localhost",
    PeerPort => "daytime(13)",
)
or die "cannot connect to daytime port at
localhost";
while ( <$remote> ) { print }
```

When you run this program, you should get something back that looks like this:

Wed May 14 08:40:46 MDT 1997

Here are what those parameters to the new constructor mean:

- **Proto** - This is which protocol to use. In this case, the socket handle returned will be connected to a TCP socket, because we want a stream-oriented connection, that is, one that acts pretty much like a plain old file. Not all sockets are this of this type. For example, the UDP protocol can be used to make a datagram socket, used for message-passing.
- **PeerAddr** - This is the name or Internet address of the remote host the server is running on. We could have specified a longer name like "www.perl.com", or an address like "204.148.40.9". For demonstration purposes, we've used the special hostname "localhost", which should always mean the current machine you're running on. The corresponding Internet address for localhost is "127.1", if you'd rather use that.
- **PeerPort** - This is the service name or port number we'd like to connect to. We could have gotten away with using just "daytime" on systems with a well-configured system services file, [FOOTNOTE: The system services file is in /etc/services under Unix] but just in case, we've specified the port number (13) in parentheses. Using just the number

would also have worked, but constant numbers make careful programmers nervous.

Notice how the return value from the new constructor is used as a filehandle in the while loop? That's what's called an indirect filehandle, a scalar variable containing a filehandle. You can use it the same way you would a normal filehandle. For example, you can read one line from it this way:

```
$line = <$handle>;
```

all remaining lines from is this way:

```
@lines = <$handle>;
```

and send a line of data to it this way:

```
print $handle "some data\n";
```

A Webget Client

Here's a simple client that takes a remote host to fetch a document from, and then a list of documents to get from that host. This is a more interesting client than the previous one because it first sends something to the server before fetching the server's response.

ListSTAR®

www.liststar.com

The most flexible email processing system available. Easily create mailing lists or email-on-demand services. Use built in rules and/or AppleScript/AppleEvents to handle any email task, no matter how simple or complex. Demo available.

MacRADIUS™

www.macradius.com

The easiest to use RADIUS server available. The groups feature allows you to make changes for a large number of users in one easy step. Enabling or disabling access for a user or a group of users is a one click operation, without having to stop the server. Support for AppleScript/AppleEvents makes it easy to control MacRADIUS. Demo available.

SimpleText Filter

Plug-in for EIMS*

www.mcfsoftware.com/stf/

Easy to use header and content filtering. Scan incoming messages for sequences of text, digits, etc. Base64 messages are decoded so you can check for content despite attempts to hide the text. Demo available.

Auto Reply

Plug-in for EIMS*

www.mcfsoftware.com/ar/

This plug-in allows you to easily set up auto-reply messages for users. Addresses that have been sent auto-reply messages are tracked, preventing auto-reply message loops. Demo available.

Address List Sorter

www.mcfsoftware.com/als/

Fast and powerful utility for sorting and cleaning email lists. Sort email address lists in alphabetical or domain order, remove duplicates and improperly formed addresses. Demo available.

*EIMS—Eudora Internet Mail Server (www.eudora.co.nz)

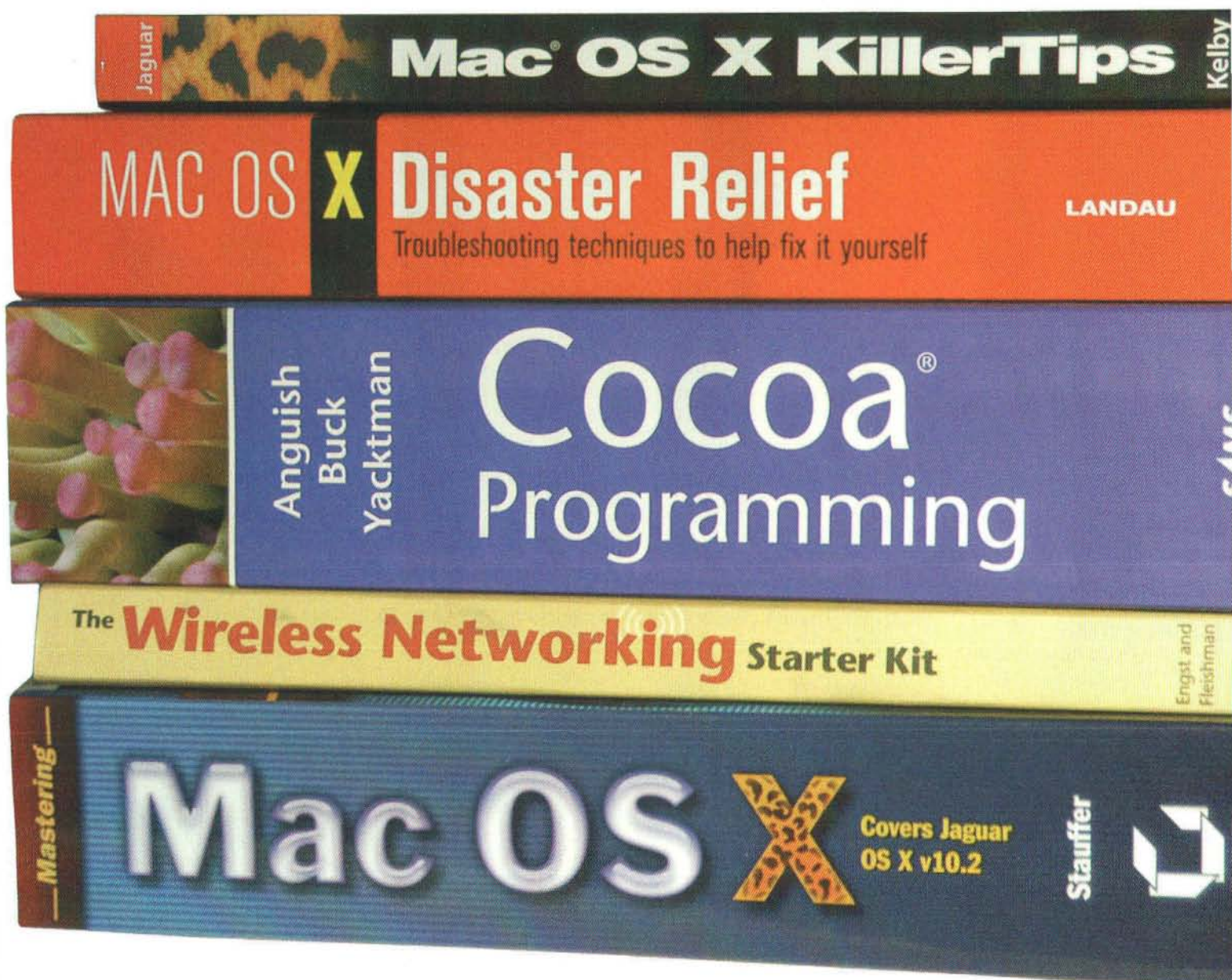


...*simply*
dependably
engineered

www.mcfsoftware.com

*You don't have to know all the answers,
if you know where to find them...*

**DevDepot has the latest references,
tutorials, tips and tricks books for Mac OS X**



Mac OS X Killer Tips by Scott Kelby **\$19.95** Have you ever had a buddy ask if you "want to know an easy way to do that?" They lean over your keyboard, click a few keys and you learn an undocumented keyboard shortcut, a cool hack, a hidden system option, or just a cleaner way to get the job done. With Mac OS X, most of our favorite tricks are gone – it's a whole new game. This 288 page book is cover to cover tips and tricks for getting the most out of Mac OS X (Jaguar).

Mac OS X Disaster Relief by Ted Landau **\$19.95** The unexpected will happen, sometimes you'll need more than a manual. _MacOS X Disaster Relief_ is the first trouble shooting guide to Mac OS X, written by Ted Landau (founder of MacFixIt). This book includes information about Mac OS X's invisible files, third party troubleshooting tools, the smart way to install (or re-install) Mac OS X – plus tips, tricks, and troubleshooting techniques.

Cocoa Programming by Scott Anguish, Erik Buck & Donald Yacktmann **\$39.95** Over 1200 pages the most comprehensive Mac programming book available for the new Cocoa API. Not for beginners, an essential reference for the serious Mac programmer.

The Wireless Networking Starter Kit by Adam Engst & Glenn Fleishman **\$24.95** Practical advice and step-by-step instructions for dozens of common tasks, troubleshooting advice to help you out of sticky situations, and coverage of Mac and Windows implementations of 802.11b, 802.11a, and 802.11g, and other technologies.

Mastering Mac OS X by Todd Stauffer **\$26.95** Mac OS X is the most significant change in the Macintosh operating system since day one – you will have questions. Questions about troubleshooting, security, networking, data recovery, fonts, AppleScript, maintenance, or other subjects. There are new technologies you've never seen before, like Darwin, shells, the command line, handwriting recognition, permissions, administration, firewalls, and others. At some point, you will want this reference.

Mac OS X for Unix Geeks by Brian Jepson, Ernest E. Rothman **\$16.95** Mac OS X has a BSD core. It's a heady feeling to have the power and flexibility of UNIX on Macintosh, but if you are a UNIX guy you'll find there are some subtle differences. This book is a solid reference, to help you make the switch. It shows you everything from your familiar user shells and directory services to building applications and system management – and how each is done under OS X 10.2 (Jaguar).

Mac OS X Hacks by Rael Dornfest, Kevin Hemenway **\$16.95** Mac OS X presents a unique opportunity for combining traditional Unix hacking and Mac OS know-how. This book goes beyond the man pages, pulling the best tips, tricks, and tools from the Mac power users and Unix hackers themselves.

Cocoa Recipes for Mac OS X by Bill Cheeseman **\$29.95** A step by step how-to on building an application with Apple's Cocoa framework. The 21 individual recipes, over 750 pages, walk you through the process of building a single, industrial strength application – while illustrating essential topics like handling menus, windows, data storage, user preferences, drag and drop, tables, undo/redo, and more. Want to get started with Cocoa? This is the book.

DEV DEPOT®

www.devdepot.com

877-DEPOT-NOW

```
#!/usr/bin/perl -w
use IO::Socket;
unless (@ARGV > 1) { die "usage: $0 host document ..." }
$host = shift(@ARGV);
$EOL = "\015\012";
$BLANK = $EOL x 2;
foreach $document ( @ARGV ) {
    $remote = IO::Socket::INET->new( Proto => "tcp",
                                     PeerAddr => $host,
                                     PeerPort => "http(80)",
                                     );
    unless ($remote) { die "cannot connect to http daemon on $host" }
    $remote->autoflush(1);
    print $remote "GET $document HTTP/1.0" . $BLANK;
    while ( <$remote> ) { print }
    close $remote;
}
}
```

The web server handing the "http" service, which is assumed to be at its standard port, number 80. If the web server you're trying to connect to is at a different port (like 1080 or 8080), you should specify as the named-parameter pair, PeerPort => 8080. The autoflush method is used on the socket because otherwise the system would buffer up the output we sent it. (If you're on a Mac, you'll also need to change every "\n" in your code that sends data over the network to be a "\015\012" instead.)

Connecting to the server is only the first part of the process: once you have the connection, you have to use the server's language. Each server on the network has its own little command language that it expects as input. The string that we send to the server starting with "GET" is in HTTP syntax. In this case, we simply request each specified document. Yes, we really are making a new connection for each document, even though it's the same host. That's the way you always used to have to speak HTTP. Recent versions of web browsers may request that the remote server leave the connection open a little while, but the server doesn't have to honor such a request.

Here's an example of running that program, which I'll call `webget`:

```
% webget www.perl.com /guanaco.html
HTTP/1.1 404 File Not Found
Date: Thu, 08 May 1997 18:02:32 GMT
Server: Apache/1.2.2b6
Connection: close
Content-type: text/html

<HEAD><TITLE>404 File Not Found</TITLE></HEAD>
<BODY><H1>File Not Found</H1>
The requested URL /guanaco.html was not found on this
server.<P>
</BODY>
```

Ok, so that's not very interesting, because it didn't find that particular document. But a long response wouldn't have fit on this page.

For a more fully-featured version of this program, you should look to the `http-request` program included with the LWP modules from CPAN.

Interactive Client with IO::Socket

Well, that's all fine if you want to send one command and get one answer, but what about setting up something

fully interactive, somewhat like the way telnet works? That way you can type a line, get the answer, type a line, get the answer, etc.

This client is more complicated than the two we've done so far, but if you're on a system that supports the powerful fork call, the solution isn't that rough. Once you've made the connection to whatever service you'd like to chat with, call fork to clone your process. Each of these two identical process has a very simple job to do: the parent copies everything from the socket to standard output, while the child simultaneously copies everything from standard input to the socket. To accomplish the same thing using just one process would be much harder, because it's easier to code two processes to do one thing than it is to code one process to do two things. (This keep-it-simple principle a cornerstones of the Unix philosophy, and good software engineering as well, which is probably why it's spread to other systems.)

Here's the code:

```
#!/usr/bin/perl -w
use strict;
use IO::Socket;
my ($host, $port, $skidpid, $handle, $line);

unless (@ARGV == 2) { die "usage: $0 host port" }
($host, $port) = @ARGV;

# create a tcp connection to the specified host and port
$handle = IO::Socket::INET->new(Proto => "tcp",
                                PeerAddr => $host,
                                PeerPort => $port)
    or die "can't connect to port $port on $host: $!";
$handle->autoflush(1); # so output gets there right away
print STDERR "[Connected to $host:$port]\n";

# split the program into two processes, identical twins
die "can't fork: $!" unless defined($skidpid = fork());

# the if() block runs only in the parent process
if ($skidpid) {
    # copy the socket to standard output
    while (defined ($line = <$handle>)) {
        print STDOUT $line;
    }
    kill("TERM", $skidpid); # send SIGTERM to child
}

# the else() block runs only in the child process
else {
    # copy standard input to the socket
    while (defined ($line = <STDIN>)) {
        print $handle $line;
    }
}
```

The kill function in the parent's if block is there to send a signal to our child process (current running in the else block) as soon as the remote server has closed its end of the connection.

If the remote server sends data a byte at time, and you need that data immediately without waiting for a newline (which might not happen), you may wish to replace the while loop in the parent with the following:

```
my $byte;
while (sysread($handle, $byte, 1) == 1) {
    print STDOUT $byte;
}
```

Making a system call for each byte you want to read is not very efficient (to put it mildly) but is the simplest to explain and works reasonably well.

TCP SERVERS WITH IO::Socket

As always, setting up a server is little bit more involved than running a client. The model is that the server creates a special kind of socket that does nothing but listen on a particular port for incoming connections. It does this by calling the `IO::Socket::INET->new()` method with slightly different arguments than the client did.

- **Proto** - This is which protocol to use. Like our clients, we'll still specify "tcp" here.
- **LocalPort** - We specify a local port in the `LocalPort` argument, which we didn't do for the client. This is service name or port number for which you want to be the server. (Under Unix, ports under 1024 are restricted to the superuser.) In our sample, we'll use port 9000, but you can use any port that's not currently in use on your system. If you try to use one already in used, you'll get an "Address already in use" message. Under Unix, the `netstat -a` command will show which services current have servers.
- **Listen** - The `Listen` parameter is set to the maximum number of pending connections we can accept until we turn away incoming clients. Think of it as a call-waiting queue for your telephone. The low-level `Socket` module has a special symbol for the system maximum, which is `SOMAXCONN`.
- **Reuse** - The `Reuse` parameter is needed so that we restart our server manually without waiting a few minutes to allow system buffers to clear out.

Once the generic server socket has been created using the parameters listed above, the server then waits for a new client to connect to it. The server blocks in the `accept` method, which

eventually accepts a bidirectional connection from the remote client. (Make sure to autoflush this handle to circumvent buffering.)

To add to user-friendliness, our server prompts the user for commands. Most servers don't do this. Because of the prompt without a newline, you'll have to use the `sysread` variant of the interactive client above.

This server accepts one of five different commands, sending output back to the client. Note that unlike most network servers, this one only handles one incoming client at a time.

Here's the code.

```
#!/usr/bin/perl -w
use IO::Socket;
use Net::hostent; # for OO version of gethostbyaddr

$PORT = 9000; # pick something not in use

$server = IO::Socket::INET->new( Proto => 'tcp',
                                LocalPort => $PORT,
                                Listen=> SOMAXCONN,
                                Reuse => 1);

die "can't setup server" unless $server;
print "[Server $0 accepting clients]\n";

while ($client = $server->accept()) {
    $client->autoflush(1);
    print $client "Welcome to $0; type help for command list.\n";
    $hostinfo = gethostbyaddr($client->peeraddr);
    printf "[Connect from %s]\n", $hostinfo->name ||
    $client->peerhost;
    print $client "Command? ";
    while ( <$client> ) {
        next unless /\S/; # blank line
        if (/quit|exit/i) { last; }
        elsif (/date|time/i) { printf $client "%s\n", scalar
        localtime; }
        elsif (/who/i ) { print $client `who 2>&l`; }
        elsif (/cookie/i ) { print $client
        `/usr/games/fortune 2>&l`; }
        elsif (/motd/i) { print $client `cat /etc/motd
        2>&l`; }
        else {
            print $client "Commands: quit date who cookie
            motd\n";
        }
    }
}
```



Valentina

Object-Relational SQL Database

The fastest database engine
for MacOS/Windows

It operates 100's and sometimes
a 1000 times faster than other systems

www.paradigmasoft.com

Hosted by macserve.net

Download full featured evaluation version

```

} continue {
    print $client "Command? ";
}
close $client;
}

```

UDP: MESSAGE PASSING

Another kind of client-server setup is one that uses not connections, but messages. UDP communications involve much lower overhead but also provide less reliability, as there are no promises that messages will arrive at all, let alone in order and unmangled. Still, UDP offers some advantages over TCP, including being able to "broadcast" or "multicast" to a whole bunch of destination hosts at once (usually on your local subnet). If you find yourself overly concerned about reliability and start building checks into your message system, then you probably should use just TCP to start with.

Note that UDP datagrams are not a bytestream and should not be treated as such. This makes using I/O mechanisms with internal buffering like `stdio` (i.e. `print()` and friends) especially cumbersome. Use `syswrite()`, or better `send()`, like in the example below.

Here's a UDP program similar to the sample Internet TCP client given earlier. However, instead of checking one host at a time, the UDP version will check many of them asynchronously by simulating a multicast and then using `select()` to do a timed-out wait for I/O. To do something similar with TCP, you'd have to use a different socket handle for each host.

```

#!/usr/bin/perl -w
use strict;
use Socket;
use Sys::Hostname;

my ( $count, $hisiaddr, $hispaddr, $hstime,
    $host, $iaddr, $paddr, $port, $proto,
    $rin, $rout, $rtime, $SECS_of_70_YEARS );

$SECS_of_70_YEARS = 2208988800;

$iaddr = gethostbyname(hostname());
$proto = getprotobyname('udp');
$port = getservbyname('time', 'udp');
$paddr = sockaddr_in(0, $iaddr); # 0 means let kernel pick

socket(SOCKET, PF_INET, SOCK_DGRAM, $proto) || die "socket:
$!";
bind(SOCKET, $paddr) || die "bind: $!";

$| = 1;
printf "%-12s %8s %s\n", "localhost", 0, scalar localtime time;
$count = 0;
for $host (@ARGV) {
    $count++;
    $hisiaddr = inet_aton($host) || die "unknown host";
    $hispaddr = sockaddr_in($port, $hisiaddr);
    defined(send(SOCKET, 0, 0, $hispaddr)) || die "send
$host: $!";
}

$rin = "";
vec($rin, fileno(SOCKET), 1) = 1;

# timeout after 10.0 seconds
while ($count && select($rout = $rin, undef, undef, 10.0)) {
    $rtime = "";
    ($hispaddr = recv(SOCKET, $rtime, 4, 0)) || die "recv: $!";
    ($port, $hisiaddr) = sockaddr_in($hispaddr);

```

```

$host = gethostbyaddr($hisiaddr, AF_INET);
$hstime = unpack("N", $rtime) - $SECS_of_70_YEARS;
printf "%-12s ", $host;
printf "%8d %s\n", $hstime - time, scalar
localtime($hstime);
$count--;
}

```

Note that this example does not include any retries and may consequently fail to contact a reachable host. The most prominent reason for this is congestion of the queues on the sending host if the number of list of hosts to contact is sufficiently large.

SysV IPC

While System V IPC isn't so widely used as sockets, it still has some interesting uses. You can't, however, effectively use SysV IPC or Berkeley `mmap()` to have shared memory so as to share a variable amongst several processes. That's because Perl would reallocate your string when you weren't wanting it to.

Here's a small example showing shared memory usage.

```

use IPC::SysV qw(IPC_PRIVATE IPC_RMID S_IRWXU);

$size = 2000;
$id = shmget(IPC_PRIVATE, $size, S_IRWXU) || die "$!";
print "shm key $id\n";

$message = "Message #1";
shmwrite($id, $message, 0, 60) || die "$!";
print "wrote: '$message'\n";
shmread($id, $buff, 0, 60) || die "$!";
print "read : '$buff'\n";

# the buffer of shmread is zero-character end-padded.
substr($buff, index($buff, "\0")) = "";
print "un" unless $buff eq $message;
print "swell\n";

print "deleting shm $id\n";
shmctl($id, IPC_RMID, 0) || die "$!";

```

Here's an example of a semaphore:

```

use IPC::SysV qw(IPC_CREAT);

$IPC_KEY = 1234;
$id = semget($IPC_KEY, 10, 0666 | IPC_CREAT) || die "$!";
print "shm key $id\n";

```

Put this code in a separate file to be run in more than one process. Call the file take:

```

# create a semaphore

$IPC_KEY = 1234;
$id = semget($IPC_KEY, 0, 0);
die if !defined($id);

$semnum = 0;
$semflag = 0;

# 'take' semaphore
# wait for semaphore to be zero
$semop = 0;
$opstring1 = pack("s!s!s!", $semnum, $semop, $semflag);

# Increment the semaphore count
$semop = 1;

```

```
$opstring2 = pack("s!s!s!", $semnum, $semop, $semflag);
$opstring = $opstring1 . $opstring2;

semop($id,$opstring) || die "$!";
```

Put this code in a separate file to be run in more than one process. Call this file give:

```
# 'give' the semaphore
# run this in the original process and you will see
# that the second process continues
```

```
$IPC_KEY = 1234;
$id = semget($IPC_KEY, 0, 0);
die if !defined($id);
```

```
$semnum = 0;
$semflag = 0;
```

```
# Decrement the semaphore count
$semop = -1;
$opstring = pack("s!s!s!", $semnum, $semop, $semflag);

semop($id,$opstring) || die "$!";
```

The SysV IPC code above was written long ago, and it's definitely clunky looking. For a more modern look, see the IPC::SysV module which is included with Perl starting from Perl 5.005.

A small example demonstrating SysV message queues:

```
use IPC::SysV qw(IPC_PRIVATE IPC_RMID IPC_CREAT S_IRWXU);

my $id = msgget(IPC_PRIVATE, IPC_CREAT | S_IRWXU);

my $sent = "message";
my $type = 1234;
```

```
my $rcvd;
my $type_rcvd;

if (defined $id) {
    if (msgsnd($id, pack("l! a*", $type_sent, $sent), 0)) {
        if (msgrcv($id, $rcvd, 60, 0, 0)) {
            ($type_rcvd, $rcvd) = unpack("l! a*", $rcvd);
            if ($rcvd eq $sent) {
                print "okay\n";
            } else {
                print "not okay\n";
            }
        } else {
            die "# msgrcv failed\n";
        }
    } else {
        die "# msgsnd failed\n";
    }
    msgctl($id, IPC_RMID, 0) || die "# msgctl failed: $!\n";
} else {
    die "# msgget failed\n";
}
```

Most of these routines quietly but politely return undef when they fail instead of causing your program to die right then and there due to an uncaught exception. (Actually, some of the new *Socket* conversion functions croak() on bad arguments.) It is therefore essential to check return values from these functions. Always begin your socket programs this way for optimal success, and don't forget to add **-T** taint checking flag to the **#!** line for servers:

```
#!/usr/bin/perl -Tw
use strict;
use sigtrap;
use Socket;
```

It's 3AM on Sunday morning...



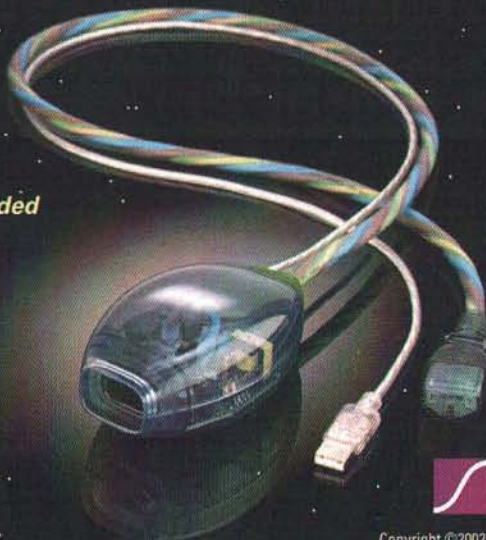
IS THE SERVER STILL RUNNING?

**End the
worry with**

Kick-off!

**The simple reliability
solution for Mac OS 9 and X**

- + **System-level Crash Detection**
patented hardware-software integration
- + **Automatic Crash Recovery**
system restart or power cycle, as needed
- + **Monitor Custom Applications**
software developer's kit and plug-ins included
- + **Scheduled Restarts**
cures memory leaks and fragmentation
- + **Restart after Power Failure**
even if shut down by your UPS
- + **Comprehensive Error Logging**
to help you track down problems
- + **Simple Installation**
just plug in AC cord and USB cable



For all these features *plus* six power outlets controllable by phone tones, schedules and scripts, consider our **PowerKey Pro 650 Admin**.

www.sophisticated.com

SOPHISTICATED CIRCUITS INC.

Copyright ©2002 Sophisticated Circuits, Inc. Kick-off! and PowerKey are registered trademarks of Sophisticated Circuits, Inc. Mac OS is a registered trademark of Apple Computer, Inc.

By Rich Morin

The Process Tree

Which processes do what

The initial design of Unix (by Ken Thompson, Dennis Ritchie, et al) was based on the idea that large jobs could be done by lots of little programs, working together. Part of the reason for this was that the available hardware had only minuscule amounts of RAM. Another part was the designers' dislike of large, monolithic programs.

Mac OS X inherits this modular approach. Most of its programs aren't "little", especially by the standards of early Unix, but there are certainly lots of them. On my freshly-booted desktop machine, I found 62:

```
% ps -ax | wc -l
62
```

The command-line options (-ax) tell **ps** to show all processes, even if they belong to another user or have no controlling terminal. Using another of **ps**'s many options, we can take a detailed look, concentrating on the commands and their parentage. Stretch a Terminal window really wide, then try:

```
% ps -axo pid,ppid,tt,user,command
PID  PPID  TT  USER  COMMAND
1      0  ??   root  /sbin/init
2      1  ??   root  /sbin/mach_init
51     1  ??   root  kextd
71     1  ??   root  update
...
539   536  std   root  login -pf rdm
540   539  std   rdm   -tsh (tsh)
1506  540  std   root  ps -axo pid ppid tt user command
674   536  p2    root  login -pf rdm
675   674  p2    rdm   -tsh (tsh)
```

The **PID** (Process ID) and **PPID** (Parent Process ID) columns allow us to guess at the "genealogy" of each process. If a process has **PPID** 1, for instance, its "parent" has **PID** 1. Note, however, that a **PPID** of 1 may also mean that the original parent process has terminated.

The **TT** ("terminal"; originally from teletype) column tells us the identity of the controlling terminal ("??" for none). Because most OSX applications have no controlling terminal, this field is insufficient to distinguish apps from background processes.

Fortunately, the **USER** (username) and **COMMAND** columns help to clear this up. Apps generally run with the username of

the logged-in user; in addition, most apps have long path names which include directories such as "**Applications**". Also, the names of BSD-derived daemons often end in "d" (e.g., **configd**, **cupsd**).

If you're seriously interested in looking at processes (e.g., to figure out which one is bogging down your system :-), you should also investigate **psstat(1)**, **top(1)**, **vm_stat(1)**. Apple's Process Viewer utility is also handy, but note that it may not report information in a manner that is consistent with the command-line tools. For example, it shows a process named "Window Manager", which **ps(1)** and **top(1)** do not list...

BACKGROUND PROCESSES

When OSX starts up, it initiates a number of background processes (commonly referred to as "daemons", after the "attendant spirits" found in Greek mythology). Because these daemons have low Process IDs, they appear at the start of the listing.

Note: Although OSX uses a 32-bit value (**pid_t**) for process IDs, it "wraps" the IDs at ~32K, so a **ps(1)** listing taken from a long-running (and/or busy) system may show other processes with low IDs. Interestingly, FreeBSD does not seem to limit IDs in this manner.

Most daemons are started up by **init(8)**, as part of the system start-up process. This accounts for the long list of processes whose **PPID** is 1. Scattered in this list, however, you may see a few processes with **PPIDs** of 2, indicating that they are children of **mach_init(8)**. Finally, some applications (e.g., **iChat**) and daemons (e.g., **loginwindow**, **WindowServer**) start up their own daemons.

Here's an annotated list of some background processes that show up on my machine. Lacking any other organizing rationale, I'll use alphabetical order. If nothing else, that will make the list easy to navigate (-):

AppleFileServer – personal file sharing server; supports AppleTalk Filing Protocol (AFP) over Internet Protocol (IP).

ATSServer – Apple Type Solution server; enables system-wide font management.

autodiskmount(8) – checks and mounts disk-based file systems.

Rich Morin has been using computers since 1970, Unix since 1983, and Mac-based Unix since 1986 (when he helped Apple create A/UX 1.0). When he isn't writing this column, Rich runs Prime Time Freeware (www.ptf.com), a publisher of books and CD-ROMs for the Free and Open Source software community. Feel free to write to Rich at rdm@ptf.com.

automount – automatically mounts and unmounts network (NFS and AFP) file systems. See **amd(8)** for information on the traditional (NFS-only) variant of this daemon.

configd – maintains dynamic configuration information about the computer and its environment (e.g. network).

coreservicesd – core services daemon.

crashreporterd – logs information about program crashes.

cron(8) – executes scheduled commands. See also **crontab(1,5)**.

cupsd(8) – Common Unix Printing System daemon. Run “**apropos cups**” to see an extensive list of man pages.

DirectoryService – directory server for Apple's Open Directory architecture.

dynamic_pager – assists the kernel with managing swap files for virtual memory.

httpd(8) – Apache hypertext transfer protocol server; may be replicated, for performance. Run “**apropos http**” to see an extensive list of man pages.

inetd(8) – internet “super-server”; listens for connections on certain internet sockets. When a connection is found on one of its sockets, it decides what service the socket corresponds to, and invokes a program to service the request.

kextd(8) – handles requests from the kernel to load kernel extensions (kexts).

loginwindow – handles miscellaneous process- and session-monitoring functions (e.g., login, logout, restart, shutdown, and restarting of the Dock and Finder).

lookupd(8) – caches directory service information (e.g., user accounts, groups, printers, e-mail aliases and distribution lists, computer names, Internet addresses).

mDNSResponder – multicast-DNS responder; advertises network services (such as AFP file sharing) provided by this computer (part of Rendezvous).

netinfod(8) – serves NetInfo information to the network. Run “**apropos netinfo**” to see an extensive list of man pages.

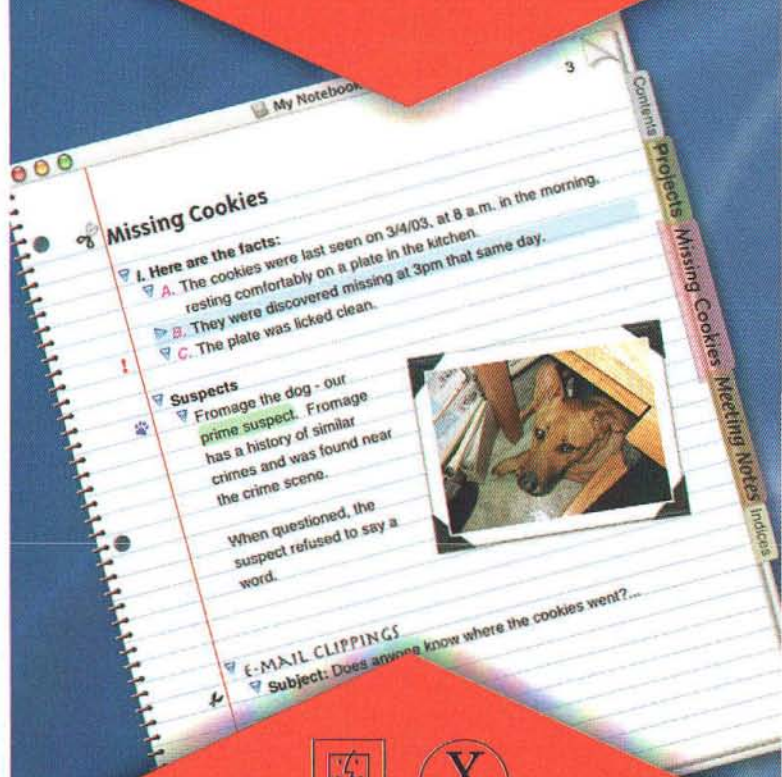
nfsiod(8) – typically, several copies of this daemon will be running on any NFS client machine, servicing asynchronous I/O requests. This daemon improves NFS performance, but it is not required for correct operation. Run “**apropos nfs**” to see an extensive list of man pages (but ignore the bogus hits for **config(5)** and **confstr(3)** :-).



NOTEBOOK™

Getting organized just got easier

- Intuitive outlining
- Customizable notebook interface
- Annotation with stickers, keywords, & highlighting
- Dynamic indexing
- Super-Find searching
- Drag-and-drop Page Bundles
- Inter-application Clipping Services
- Direct import from digital cameras
- Rich image handling
- Media Frames
- Export to XML
- Online help



circusponies.com

ntpd(8:FreeBSD) – Network Time Protocol (NTP) daemon; sets and maintains the system time of day in synchronism with Internet standard time servers.

pbs – pasteboard server (similar to the Clipboard in Mac OS 9); enables the exchange of data between applications. It is also the data-transfer mechanism used in dragging operations. (started by `loginwindow`).

SecurityServer – oversees system authorization, authentication, and keychain access.

slpd – Service Location Protocol (SLP) responder; advertises network services (such as AFP file sharing) provided by this computer.

sshd(8) – with `ssh(1)`, replaces `rlogin(1)` and `rsh(1)`, providing secure encrypted communications between two untrusted hosts over an insecure network. Run “`apropos ssh`” to see an extensive list of man pages.

syslogd(8) – reads and logs messages to the system console, log files, other machines, and/or users as specified by `syslog.conf(5)`, its configuration file. See also `syslog(3)`.

SystemUIServer – displays items on the right-hand end of the menu bar; loads menu and dock extras as plug-ins (started by `WindowServer`).

update(8) – helps protect the integrity of disk volumes by flushing volatile cached file system data to disk at thirty second intervals.

WindowServer – responsible for rudimentary screen displays, window compositing and management, event routing, and cursor management. It coordinates low-level windowing behavior and enforces a fundamental uniformity in what appears on the screen.

INTERACTIVE PROCESSES

Most apps are children of the `WindowServer` daemon. Their names should be quite familiar: `Dock`, `Finder`, `iChat`, `Preview`, `Terminal`. As noted above, some apps start up their own daemons. For example, `iChat` starts up `iChatAgent` (probably to handle communications).

The lineage for command-line programs is a bit more complex. Assuming that you're using `Terminal`, it should look something like this:

PID	PPID	TT	USER	COMMAND
173	1	??	rdm	.../WindowServer ...
536	173	??	rdm	.../Terminal ...
539	536	p1	root	login -pf rdm
540	539	p1	rdm	-tcsch (tcsch)
873	540	p1	root	top

As these lines show, `init(8)` started up `WindowServer`, which started up the `Terminal`. Then, to generate an interactive shell window, `Terminal` started up `login`, which started up `tcsch(1)`. Finally, I ran `top(1)` from the command line.

Careful Reader will notice that the username for `login` and `top` is `root`, rather than `rdm`. Because these processes need to do some things which a normal user is not allowed to do, they have been created as `setuid(2)` executables:

```
% ls -l /usr/bin/{login,top}
-r-sr-xr-x 1 root wheel ... /usr/bin/login
-r-sr-xr-x 1 root wheel ... /usr/bin/top
```

FURTHER READING

In a traditional BSD system, the first place to look for information on commands and daemons would be the manual. Unfortunately, many OSX commands and daemons do not have accompanying manual pages. Even when man pages are present, they may not be up-to-date, let alone customized to reflect changes which Apple has made, other documentation it has developed, etc.

A comprehensive attack on this problem has been suggested by parties inside and outside of Apple, but (AFAIK) no significant effort has been made as yet. If you would like better manual pages, let ADC (etc.) know!

In the meanwhile, here are some places where I found useful information for this section:

<http://www.westwind.com/reference/OS-X/background-processes.html>

http://developer.apple.com/techpubs/macosx/Essentials/SystemOverview/BootingLogin/chapter_4_section_5.html

HelpLogicTM

The Help Authoring Solution for Mac Developers



Easily create help systems for your software applications & web sites from a single source.

Save time with the integrated Workshop, TOC Builder, HTML Editor & Page Templates to quickly generate Apple Help, Web-based Help, UniHelp, PDF & more.

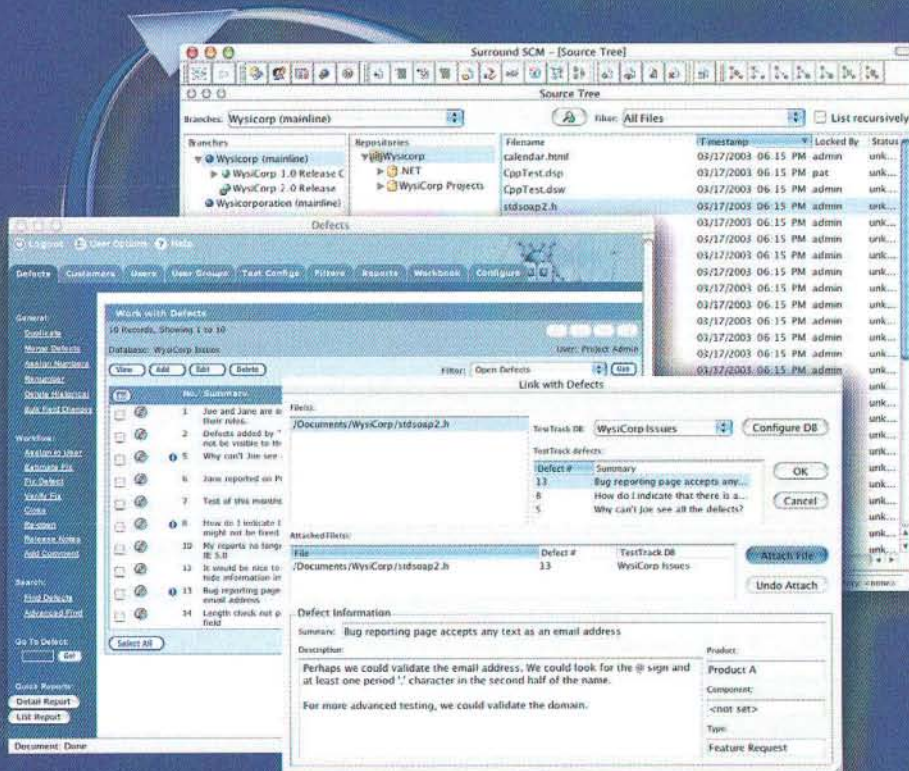
SNEAK PREVIEW
www.ebutterfly.com

electric butterfly

Complete Source Control and Defect Management

Seapine Software™
changing the world
of software development

for Mac OS X



Effective source code control and defect tracking require powerful, flexible, and easy-to-use tools—Surround SCM and TestTrack Pro

- Complete source code control with private workspaces, automatic merging, role-based security, and more
- Comprehensive defect management — track bug reports and change requests, define workflow, customize fields
- Fast and secure remote access to your source files and defects — work from anywhere
- Advanced branching simplifies managing multiple versions of your products
- Link code changes with defects and change requests — know who changed what, when, and why
- Scalable and reliable cross-platform, client/server solutions support Mac OS X, Windows, Linux, and Solaris
- Exchange data using XML and ODBC, extend and automate with SOAP support
- Licenses priced to fit your budget

Seapine Software Product Lifecycle Management
Award winning, easy-to-use software development tools

Seapine
Surround SCM

Seapine
TestTrack PRO



Download Surround SCM
and TestTrack Pro at
www.seapine.com
or call 1-888-683-6456

all product names listed herein are registered trademarks of their respective owners. All rights reserved.



By John A. Vink

Puzzle 2: What's on the menu?

Try to solve this programming puzzle before the score drops to zero. The puzzle is presented as a discussion among engineers in a chat room. As the engineers gather information and make suggestions, see if you can find the solution. If you solve the problem before you get to the end of the puzzle, you get the score in the left hand column.

SDK: I love the monotony of a steady paycheck.

Argus: Tell me about it, SDK.

100 ChrisE: OK, under what circumstances does GetNewMBar return nil?

95 BMA: No MBar resource?

ChrisE: It's there.

90 SDK: Out of memory? Like, really out?

ChrisE: Mac OS X

SDK: Heh, mine runs out of memory all the time.

SDK: I can tell because it gets slower and console gets really talkative.

85 BMA: What's ResError and MemError have to say about it?

80 ChrisE: ResError() is 0; MemError() is 0.

BMA: Your computrons are leaking out of the bottom of the case, you need to plug it up with silicone fittings.

BMA: Is this Tuttle's machine? It's cursed, I tells you!

75 Argus: Resource not linked in?

70 Argus: Can you use GetResource on the MBar?

ChrisE: Whaddya mean "not linked in"?

Argus: What are you really trying to do? Is the resource really available to your app?

BMA: When you said "it's there", did you look in the executable or in the project to verify?

65 ChrisE: OK, I have one project, two targets. The targets build slightly different ways, but with identical source, resources, etc.

ChrisE: One target builds and runs fine.

60 ChrisE: The other (mostly identical) target gets nil from GetNewMBar.

BMA: You got Resorcerer handy?

55 KeithS: Maybe one target requires a MDEF that isn't present in the other?

Pie R. Square: Why is ResError() 0?

50 ChrisE: I DeRez the .rsrc files in both packages and they're identical.

BMA: Bit for bit?

45 Argus: Throw in a GetResource('MBar', x) and see if you get a resource back.

Pie R. Square: They link to the same set of libraries?

ChrisE: OK, what's hex for MBar? I'm in gdb.

JAV: BaseMaster says: 4D424152

John A. Vink (aka "JAV") is a software engineer for Apple Computer, Inc. He takes a break in the summer to follow INXS on tour. His pals are a bunch of smart friends scattered throughout the Mac programming landscape. You can contact him at vink@apple.com. (The pals reference makes sense with the "John And Pals' Puzzle Page" title)

BMA: For the record, print/x "MBAR" in gdb does what you want.

40 BMA: Do you load any other resources before loading the MBAR?

ChrisE: Nope.

35 KeithS: `gdb ; br GetMenuBar(); call (int) GetResource ('MBAR', 128)` and see if it works.

ChrisE: `GetResource` returns 0.

Pic R. Square: And `ResErr()` is 0?

ChrisE: `ResError` is 0.

30 Argus: `ResError` is volatile and can get reset quickly.

25 BMA: Your plist is hosed and your resource file is not being opened at all.

BMA: Diff the plists.

20 KeithS: call `(void) PrintResourceChain(1)`

BMA: Do the resource files match the app names?

15 ChrisE: Plists differ only in the name of the app, but in the one that's broken, the plist name differs from the name of its app.

BMA: There you go.

ChrisE: In the one that works they match.

ChrisE: Bravo, guys. JAV, write this up.

ChrisE: `PrintResourceChain` shows my .rsrc isn't there, probably because the plist has the wrong name.

10 KeithS: So `CFBundle` is failing to open your resource file because it's got your executable wrong.

BMA: Well, in your case the file itself has the wrong name, but yes.

5 ChrisE: The file has the same name as the executable, but the plist entry is different. I had changed the name in the target editor but an old name persisted in the plist.

BMA: Oh I see, I got it backwards.

ChrisE: Executes correctly now.

ChrisE: Thanks! Score:

85-100 That was pretty good. 65-80 You must have called a friend on the inside. 45-60 You've been carbonizing your apps like a good developer. 25-40 You see, `GetMenuResource` is a classic Mac OS API.

Score:

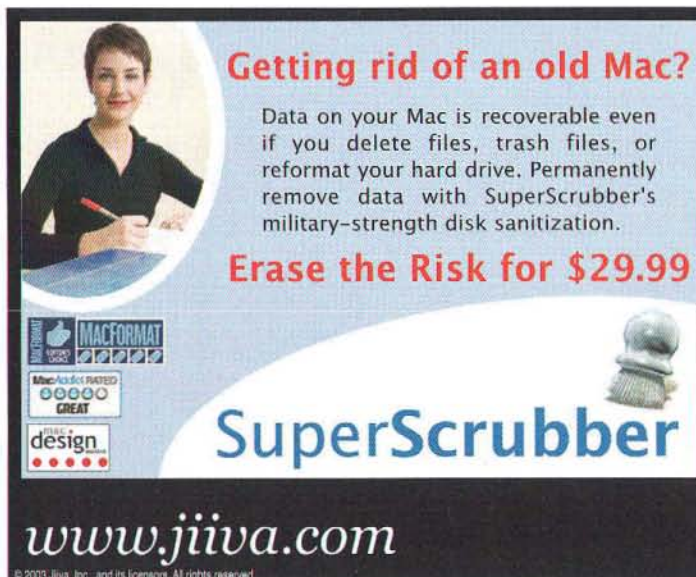
85-100 That was pretty good.

65-80 You must have called a friend on the inside.

45-60 You've been carbonizing your apps like a good developer.

25-40 You see, `GetMenuResource` is a classic Mac OS API.

0-20 Stick to Cocoa and command line apps.



Getting rid of an old Mac?

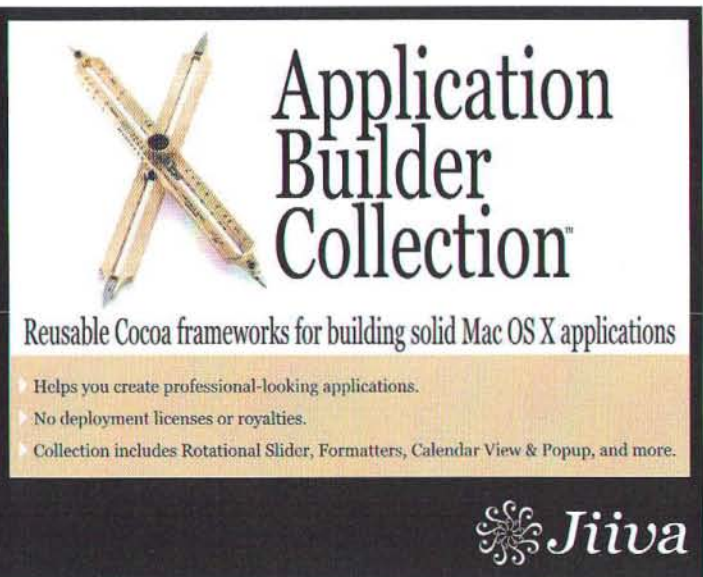
Data on your Mac is recoverable even if you delete files, trash files, or reformat your hard drive. Permanently remove data with SuperScrubber's military-strength disk sanitization.

Erase the Risk for \$29.99

SuperScrubber

www.jiiva.com

© 2003 Jiiva, Inc. and its licensors. All rights reserved.



Application Builder Collection

Reusable Cocoa frameworks for building solid Mac OS X applications

- Helps you create professional-looking applications.
- No deployment licenses or royalties.
- Collection includes Rotational Slider, Formatters, Calendar View & Popup, and more.

Jiiva

By John C. Daub

Apple's Developer Toolchain

Or, how I learned to appreciate Objective C

Mac OS X has been publicly available for over two years. Many others and myself have come to appreciate the "native" offerings that Mac OS X brings, such as increased stability and true multitasking. But one thing that still boggles my mind is how many Mac OS software developers still balk at the "native" Mac OS X software developer toolchain; by that I mean: Cocoa, Project Builder, Interface Builder, and Objective C. I have found this toolchain greatly increases my productivity over my previous toolchain, and it pains me whenever I find a Mac OS software developer that simply refuses to give it a try. I admit I too had "bracket shock" when I first looked at Objective C source code, but I'm glad I didn't let that scare me away because I've found a way to develop software faster and better than the way I was previously developing.

COCOA

Cocoa is the name for Apple's native object oriented application development frameworks. These days it's safe to say Cocoa's chief competitor is PowerPlant, Metrowerks' object oriented application development framework. Cocoa and PowerPlant work to accomplish the same goal: wrap up the procedures of the operating system and present them to the developer in an object oriented manner, along with utility and helper classes that solve common programming problems to facilitate faster application development. I believe Cocoa does it better, and I think that's due in part to the maturity of Cocoa as an API. I've found Cocoa's abilities to be very broad and very deep, providing me with just about everything I need apart from whatever specifics there are to my application's logic. As an example, I wanted to hack up a small application to help me keep my notes. I started in PowerPlant and after two weeks was greatly frustrated at how much work I had done and how little I had accomplished because in that time I was still struggling to get the foundation for the application established. I had just finished going through the *Learning Cocoa* book and decided to try using Cocoa to hack up my application. Even with my inexperience, within a day I had gotten further working in Cocoa

than I did in PowerPlant, and I believe this was due to the breadth and depth of the API that Cocoa provides.

One thing about PowerPlant's design is the "buffet style" approach it takes. PowerPlant is intentionally designed so that its classes can be used together or separately; one is able to pick out and use what looks interesting. While I certainly find merit to this approach, in my years of using PowerPlant I've also found myself occasionally frustrated by this approach. When I worked for Pervasive Software developing Tango 2000 I used the MacApp framework, which is more of an "all or nothing" framework as it has a common base class which most all classes inherit from: TObject. Cocoa has a similar approach with most all classes inheriting from NSObject. I actually didn't find this approach as bad as I was lead to believe, and since my MacApp experiences I've grown to actually appreciate and prefer a common "object" base class. This is because I've found myself striving to write code in more of a true object oriented manner, and it helps when everything you're working with is an object.

When I work in Cocoa I feel like I'm programming the system in an object oriented fashion, unlike Carbon, which is procedural in nature. One thing that helps make this possible is how the Objective C language and runtime is integrated into the system. I'll speak more on Objective C later in the article. But along the lines of integration is the core development applications: Interface Builder and Project Builder.

PROJECT BUILDER

Project Builder is Apple's project management tool. It's akin to the CodeWarrior IDE in terms of the problem it's providing a solution for – project management. Most Mac programmers these days have used CodeWarrior and feel that Project Builder isn't quite the tool that the CodeWarrior IDE is. I tend to agree with that assessment. There are a lot of nice things that Project Builder can do because of how it's implemented, such as how it takes advantage of the runtime environment that Mac OS X provides whereas the CodeWarrior IDE is still WaitNextEvent/Thread Manager based. I do find CodeWarrior's class browser features to be wonderful and miss them greatly when I work in Project Builder, but at least as of CodeWarrior Pro 8.3 its class browser

John C. Daub is a *MacTech Magazine* Contributing Editor, a Mac OS Software Developer for Aladdin Systems, Inc., and Grand Pooh-bah of Hsui's Shop. John resides in Austin, Texas USA with his wife, three children, two cats, and a tank full of tropical fish. He strongly believes Californians have no idea what good BBQ is. Thanx to Adriaan Tijsseling for the article review. You can reach John via email at hsui@hsui.com.

didn't support Objective C (heck, the function popup still doesn't grok Objective C). But then I find Project Builder's integration with API documentation very handy. I find myself preferring Metrowerks' compilers to gcc and that the MSL C++ libraries are an excellent piece of work. But then tool integration like the CodeWarrior IDE and Constructor just isn't there like it is with Project Builder and Interface Builder.

Project Builder is certainly a more than usable product, but it does have a ways to go, especially if it wants to bring the die-hard CodeWarriors into the fold. As I revise my writing of this article, WWDC 2003 just ended and at the show Apple announced Xcode, their new integrated development environment (see Dave Mark's guided tour of Xcode elsewhere in this issue). I believe Project Builder is presently the weakest link in the Apple toolchain, but the features and promises that Xcode makes certainly raise the bar for Mac developer tools. Let's hope Xcode can deliver on those promises and that the competition between Apple and Metrowerks leads only to good things for Mac developers and ultimately Mac users. However, at present we still have to work with Project Builder and it remains the weakest link in the chain, but it doesn't bother me that much because I find Interface Builder to be one of the stronger links in the chain.

INTERFACE BUILDER

In my experience, I have found Interface Builder to be a hands-down better tool than Constructor. Constructor is a good and necessary tool, indispensable for PowerPlant programming. But I have found Interface Builder better suits my needs.

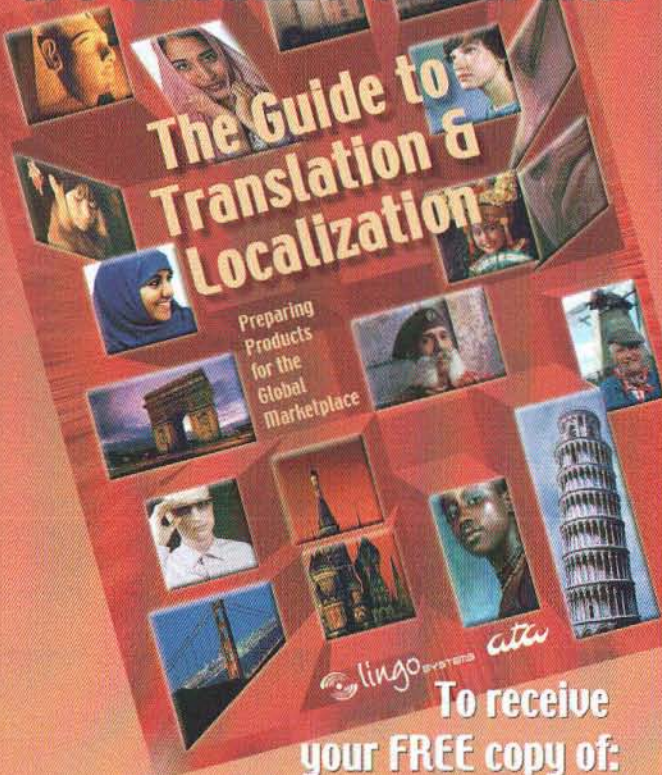
I find the whole GUI layout process works nicer in Interface Builder. When laying out widgets, there are guides to help enforce the layout dimensions and measurements from the Apple Human Interface Guidelines – I'm not wasting time and straining my eyes to count pixels. I also love how I can try out my GUI from within Interface Builder to really get a good idea about how my GUI is going to look and feel. It's a terrific tool for RAD development and prototyping, not to mention mocking up screenshots for design documentation.

Even though the application is called Interface Builder, I consider it a tool for more than just building your GUI – it's a tool to build your objects and establish relationships between those objects. In Interface Builder you can not only construct your GUI objects, but you typically will create other objects such as controllers and data sources. From within Interface Builder you can create the (sub)class, instantiate the object, and even generate the skeleton source code which Interface Builder can automatically add to your project open in Project Builder. When generating source code, Interface Builder can generate it as Java or Objective C. While Java's a good language, when I'm developing Mac OS X software I'm finding that I prefer Objective C.

OBJECTIVE C

The more I use Objective C the more I find it to be a really cool language. It looks scary at first, but it takes maybe

Classic or Cocoa Applications? We Localize Them All!



To receive
your FREE copy of:
**The Guide to Translation and
Localization – Preparing Products
for the Global Marketplace**

Call: 1-800-878-8523

Email: info@lingosys.com

Fax: 1-503-419-4873

Or visit: www.lingosys.com

**We focus on what matters most: meeting your
needs and providing excellent customer service.**

- Translation
- Desktop Publishing
- Project Management
- Localization
- Engineering
- Quality Assurance



15115 SW Sequoia Pkwy. #200 Portland, OR 97224

30 minutes to understand the basics of Objective C and perhaps a weekend of tinkering with tutorials and sample projects to really grok how the language works. If you're familiar with object oriented programming from C++ or Java, you'll find many similar concepts in Objective C; in fact, Objective C influenced Java's development. Once you understand how the language works, then you'll be ready to understand how Cocoa works.

Just because a language may not be widely popular, like Latin, there's sometimes still merit to learning and knowing that language. Objective C ultimately is C with, in my opinion, better object oriented concepts tacked onto it than C++ tacked on to C. The bracket syntax is better thought of as *message passing*. In C++ you might do this:

```
object->Method();
```

or a PowerPlant programmer might do this:

```
widget->BroadcastMessage(msg_SomeMessage, ioParam);
```

In Objective C you'd do this:

```
[object method];
```

In terms of what it functionally means, it's all the same: having an object do something. What's different is the syntax

for how messages are passed to objects: in C++ you call an object's function, in Objective C you're passing the object a message. What's also different but not apparent from looking at the code is in C++ the function and its arguments are joined together in compiled code, whereas in Objective C the message and the receiving object aren't bound together until the program is running and the message is actually sent; this is called **dynamic binding**. Dynamic binding gives Objective C a great deal of power as an object oriented programming language since at runtime a developer can vary not only the messages sent but the objects receiving those messages. Receivers and messages can be determined on the fly and be made contingent upon external factors like user input. Ultimately I think of Objective C as just another language in my toolset. But if you think about speaking languages (English, Spanish, French, etc.), the more languages you know the more you can understand, describe, and work with the world around you. Knowing more languages helps you think a little different, a little better – your horizons are broadened.

One thing I should tell you is that years ago I worked as an employee of Metrowerks as a PowerPlant engineer. As much as I adore PowerPlant, I've really come to discover the limitations with the C++ language. I believe that C++ is a great language for generic programming (templates are awesome), but I'm finding Objective C to be a better language for object oriented programming. Objective C's support for dynamic typing, dynamic binding, and messaging feel to me to lend better to the paradigms of object oriented programming. And within Apple's developer toolchain, I believe Objective C is the linchpin that makes it all possible.

Sure I miss C++ isms like using stack-based classes for cleanup, but I'm also finding that with well-crafted code I don't need those facilities when I write in Objective C. I've found constructs like categories to be very useful so that I can extend a class without having to subclass and create a new type (e.g. I can add Str255 "constructors" to NSString via a category). What I find most powerful about Objective C is how the language is dynamic and binding occurs at runtime. This has allowed me to write some really neat code that I just couldn't do in a static compile-time binding language like C++. But I haven't abandoned C++ entirely because again it's a great generic programming language. Plus if you have to write code that's cross-platform, C++ makes for a good choice. You can write your generic engine code in C++ and share that across platforms, then write your platform-specific GUI code in Cocoa (for Mac). The Objective C++ language and compiler support allows C++ and Objective C code to live together in almost perfect harmony.

SO WHAT?

Sometimes it feels strange to me to use something (Cocoa) other than what I had a hand in creating (PowerPlant).

Whistle Blower

Enterprise server monitor and restart utility
whistleblower.sentman.com

Connect to and validate the response from web servers, cgi scripts and over 23 other types of servers.

Send email, pages and perform unattended restarts via MasterSwitch or PowerKey.

Shifts make sure that the person on call when the server goes down is the one who gets the page.

68k, PPC and Carbon

Web based administration lets you check on and restart your servers from anywhere.

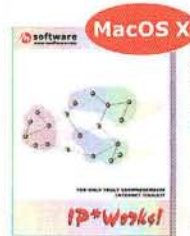
Customize your response to an outage with Apple Script.

email us at whistleblower@sentman.com

Sometimes I feel that having Objective C on my résumé won't get me as far as it would having Java on my résumé. But I've gotten over these feelings. I've come to a point in my career where I no longer care to be a religious zealot about a computing platform or a programming language or a development framework. I just want to be a zealot for creating insanely great software for Mac OS X. My users don't care if I use Carbon or Cocoa, PowerPlant or REALbasic, C++ or Objective C, CodeWarrior or Apple tools. Users just want software that provides solutions and works well, and they want it today. So when I am presented with a programming problem, I don't first pick my toolset then cram my solution into it. I now analyze the problem and work to determine what tools will be most appropriate to solve the problem. Maybe Java and the Swing libraries will be the best solution. Maybe C++ and PowerPlant or Qt will be a more appropriate tool. Or maybe quickly hacking up a Perl script will be the best tool for the job. The more variety of tools I have in my toolchest, the more power I have to pick the right tool for the job to ensure I can do the job effectively.

I have discovered that when I use Apple's developer toolchain of Cocoa, Project Builder, Interface Builder, and Objective C all coupled with the killer environment that is Mac OS X, I find I can deliver more robust solutions in less time. It may not always be my toolset of choice for a particular project, but at least since I know about and how to use the toolchain I can see the disadvantages and advantages that it has. For example, a project I'm presently working on has requirements that forced me to drop Project Builder for CodeWarrior because of some issues with the -header-mapfile option that I couldn't work around any other way. But I'm still using Cocoa, Interface Builder, and Objective C (along with C, C++, and Objective C++) in this project. Generally speaking, Apple's toolchain is becoming my toolset of choice because of the power, speed, and flexibility inherent in the system. In a few years I may feel differently. PowerPlant X is on the horizon and I know Metrowerks is working hard to maintain their lead as a tools platform. And with the release of Apple's Xcode, it's obvious that Apple isn't resting on their laurels when it comes to developer tools. I hope this competition in the developer tools space will only lead to better solutions for developers, which means better software for end users, and that keeps us all in a job.

If you haven't given Apple's developer toolchain a try, dedicate a weekend to trying it out. Set your biases aside and just give it an honest try. Even if you don't stick with it, at least you've been able to evaluate it from a more hands-on perspective. By knowing a little bit more about the toolchain, it helps you to expand your knowledge and toolset. I believe that will make you a better and more valuable software developer.



IP*Works!
**The Only Truly Comprehensive
Internet Toolkit**

- More Than 30 Components Cover All Major Internet Protocols
- Follows Exact RFC Specifications
- Market-Tested For Over 7 Years
- In Use By Almost All Fortune 500s
- Royalty-Free Pricing

IP*Works! for Mac OS X (10.0/10.1/10.2) gives you access to a host of new capabilities that will connect your applications to the Internet with unprecedented ease of use, power, and flexibility! You will be able to easily and quickly:

- *program the web: HTTP, WebForm, WebUpload*
- *call web services: SOAP, XMLp*
- *transfer files: FTP, TFTP*
- *send email: SMTP, FileMailer, HTMLMailer*
- *receive email: POP, IMAP*
- *read/post news: NNTP*
- *encode/decode: MIME, NetCode*
- *access directories: LDAP*
- *manage networks: SNMP, Whois, Ping, TraceRoute*
- *build clients and servers: IPPort, IPDaemon*
- *build packet applications: UDPPort, Multicast*
- *remote access: Telnet, Rexec, Rshell, RCP*

...and a whole lot more! - download your free trial today from www.nsoftware.com!

**IP*Works! for
Mac OS X is currently
available as a C/C++
Library (OS X Framework).
Objective-C Classes for
Cocoa and RealBasic
plugins coming soon at
www.nsoftware.com**

STAY TUNED!

By Kevin Hemenway, National Resident

Server Side Includes with Apache

Including Content within Other Content, And More

In our last column, we chatted about turning on our web server, getting more information concerning how it's been installed (like where the log and configuration files are), and then took a quick look at editing the primary control file, `/etc/httpd/httpd.conf`. After we made our changes (intended to circumvent an Evil ISP's port filter), we learned of an alternate route to restarting Apache by using the command line utility `apachectl`.

All relatively Duplo. Let's break out the Legos.

BUT FIRST, CONSIDER HOMELAND SECURITY...

By running a web server, you're inviting anyone on the 'net to stop by your computer and access files you've deemed worthy. This should be a scary thought: what about all the files you DON'T deem worthy, like development versions of your software, database files that contain customer information, or the source code to any web scripts you may be running? Even if you've got a dedicated machine solely for your web site, you've still got to wear another hat: that of security princess ("I feel preet'ly, OoOh soOO PreeTtYy!").

This isn't security like in software downloads, where you concern yourself with viruses or trojans or pirated registrations. With web server security, you've got one wide-open front door, accessible to anyone who deems to visit. Any script you run, any software you install, any service you turn on – all are more points of access for a disgruntled cracker.

While we'll talk about security when necessary, there are two books that will put more diamonds in your tiara than I ever could. Both were sent as review copies, and both have since earned welcome places on my bookshelf.

The first, *Mac OS X Maximum Security* from John Ray and William C. Ray, covers every aspect of hardening your Mac OS X installation. Broken up into three primary sections, it gets you into the mindset of thinking secure, follows up with different ways people get into systems, and then instructs on how to actually batten down the hatches. Encompassing far more than

just Apache, I'd recommend it to all readers, not just those serving web pages.

The second is much more specific to our topic: *Maximum Apache Security* by Anonymous. Psychotically comprehensive, it covers exactly (with source code) how Apache handles various bits of its own logic, as well as how it interacts with third party software like databases, scripting languages, and modules. Unlike *Mac OS X Maximum Security*, it assumes you already have a strong knowledge of how Apache works. If you do, and you're reading my columns solely for their rhythm, this is a book you should consider adding to your collection.

YEAH, YEAH, YEAH – SERVER SIDE INCLUDES, VAMANOS!

Server Side Includes (SSI) are an Apache built-in that, at its simplest, allow you to include one bit of content within another. If you're thinking variable interpolation, you've nailed it on the head. For web page design, this becomes very helpful in regards to navigation bars, headers, footers, copyright statements, or anything intended for every page. I've designed entire sites using SSI, with the content files being nothing more than semantic headers and paragraphs, and the pretty "shell" being included by Apache upon request. When a redesign occurs, modify two or three files and I'm done – the bulk of the site, containing hundreds of pages of content, remains unmodified.

That's not all SSIs can do, however. With a dash of conditionals and a smidgen of regular expressions, you've got a feature set that can quickly perform some interesting tricks for when you don't need (or want) the power of PHP or Perl (which we'll cover in future columns).

ENABLING SERVER SIDE INCLUDES: THE ECOLOGY OF A MODULE

Even though they're built into Apache, SSI's aren't enabled by default. If you recall from the last column, the easiest way to learn about a feature in Apache is to just search for it in `/etc/httpd/httpd.conf`. Open said puppy up in your favorite authenticating text editor (BBEdit for me) and do a search for the word "includes". Your first match should be:

```
LoadModule includes_module      libexec/httpd/mod_include.so
```

Kevin Hemenway, coauthor of *Mac OS X Hacks*, is better known as Morbus Iff, the creator of disobey.com, which bills itself as "content for the discontented." Publisher and developer of more home cooking than you could ever imagine (like the popular open-sourced aggregator AmphetDesk, the best-kept gaming secret Gamegrene.com, articles for Apple's Internet Developer and the O'Reilly Network, etc.), he went out twice this summer, only to scurry back inside like a disgruntled cockroach. Contact him at morbus@disobey.com.

Most of the features within Apache are controlled by modules, which can best be described as a "plugin" to the core web server code. A decent amount of modules ship with Apache already – you'll see them above and below our first search result. Here, we're loading a module named **includes_module**, which is located on the file system at `/usr/libexec/httpd/mod_include.so` (`/usr` being the Apache root directory via `HTTPD_ROOT`, see last column). When a module is enabled (indicated by no preceding comment character, #), it's ready to be configured for use.

For every **LoadModule**, there's a matching **AddModule** shortly after. To correctly enable a module within Apache, both lines need to exist uncommented. The match to the above **LoadModule** is **AddModule mod_include.c**, which you'll see in another fifty lines or so.

Any programmer can write a module to Apache, and a healthy list of third party enhancements is available at <http://modules.apache.org/>. Most modules are named **mod_SOMETHING**, like **mod_include**, **mod_php**, **mod_access**, etc, although there are occasional exceptions. If you're interested in exploring module creation for Apache, check out *Writing Apache Modules with Perl and C* (<http://www.oreilly.com/catalog/wrapmod/>).

ENABLING SERVER SIDE INCLUDES: DIRECTORY ACCESS

Our next search result for "includes", of which I've snipped only the relevant, is below. It contains the configuration for a specific directory on our machine, namely `/Library/WebServer/Documents`. Of more importance, as a concept, is the "block" or "container" – the directives within `<Directory>` only apply to the location specified. Apache has a number of block directives, which you'll see more of as the columns progress.

```
<Directory "/Library/WebServer/Documents">
# This may also be "None", "All", or any combination of
# "Indexes", "Includes", "FollowSymLinks", "ExecCGI",
# or "MultiViews".
Options Indexes FollowSymLinks MultiViews

AllowOverride None
Order allow,deny
Allow from all
</Directory>
```

The configured directory is also Apache's **DocumentRoot** – the default location from which files will be served. `http://127.0.0.1/demo/litton.html`, for instance, would reference the file located at `/Library/WebServer/Documents/demo/litton.html`. While we won't be getting into the details of the other directives above (yet), we need to add **Includes** to the **Options** line, like so:

```
Options Indexes FollowSymLinks MultiViews Includes
```

By adding **Includes**, we're instructing Apache to allow SSI's within that directory and all it's children. If we wanted to support SSI's in only a certain subdirectory, we'd need to add a new `<Directory>` block entirely. Take the example below, which ensures that only `/testbed/` (and it's children) are privileged. We don't have to specify the other directives, like **AllowOverride**, **Order**, and **Allow**, as those are inherited from the parent.

PhoneValet

Your Personal Telephone Assistant



- *Announces who is calling before you take the call.*
- *Reports to you how much time you spent on the phone and with whom.*
- *Locates the number in your phone book then dials it for you.*
- *Acts on your behalf when calls come in; selectively sends e-mails or invokes scripts.*

Talking Caller ID, Call Logging, Call-Triggered Actions and Voice Dialing on one or more lines.



Hardware, software and cables included!

www.phonevalet.com

See us at WWDC and Macworld Expo!

Developers and OEMs

Our hardware, drivers and APIs can be licensed to help you build exciting telephony solutions!

Parliant's USB Telephony Interface provides developers with the ability to dial the phone, decode Caller ID, detect dialed digits, play and record audio on and off the phone line, and get hook status.

Parliant offers high-level interfaces for Cocoa, Carbon, and C/C++ programmers to leverage our hardware without low-level USB programming.



Parliant Corporation

www.parliant.com

1-866-864-2334



```
<Directory "/Library/WebServer/Documents">
  Options Indexes FollowSymLinks MultiViews
  AllowOverride None
  Order allow,deny
  Allow from all
</Directory>

<Directory "/Library/WebServer/Documents/testbed">
  Options Includes
</Directory>
```

We've still got a little more to go before we're up and running. Let's move on.

ENABLING SERVER SIDE INCLUDES: ASSOCIATING A HANDLER

As we'll see in the second half of our article, using an SSI is a simple matter of including a special bit of code in your normal HTML. This code has to be interpreted by the SSI module, and the final bit of HTML (sans those special codes) is spit out to the browser. To interpret these codes, we need to tell Apache to associate certain files with the SSI module. This is where the last part of our configuration lies, and is our next *relevant* search result:

```
# If you want to use server side includes, or CGI outside
# ScriptAliased directories, uncomment the following lines.
#
# To use CGI scripts:
#
# AddHandler cgi-script .cgi
#
# To use server-parsed HTML files
#
# AddType text/html .shtml
# AddHandler server-parsed .shtml
```

We'll cover CGI next month, so concern yourself only with the last two lines. Both are commented (indicated by the # character that precedes them), and both help the final part of our configuration. The first command tells Apache that when a file with the extension `shtml` has been requested, the server should send a MIME type of `text/html`. This, in effect, treats all `shtml` files as if they were normal `html` (which, after parsing, they will be).

The next line is what actually associates `shtml` files with the SSI module. When someone requests one of these files, it will be "handled" by the `server-parsed` extension of Apache. When the handler is done, the completed results will be sent as a `text/html` file to the requesting user-agent (ie. your visitor's browser).

To finally enable SSIs, uncomment these last two lines, then restart the Apache web server (either through the *Sharing System Preference*, or with `sudo apachectl restart`). Once Apache has restarted successfully, we can finally move on to something demonstrative.

PLAYING WITH SERVER SIDE INCLUDES: OUR FIRST ATTEMPT

We're now going to create two files. The first will be the "shell" that includes some outside data, and the second file will be the outside data itself. Open your favorite text editor, add the following into a document called `index.shtml`, and save that file into `/Library/WebServer/Documents`:

```
<html>
<head>
  <title>Quote Selector</title>
```

```
</head>
<body>

<h1>Quote Selector</h1>

<blockquote>
  <!--#include virtual="quote.shtml" -->
</blockquote>

</body>
</html>
```

Note that our filename has an extension of `shtml` (`index.shtml`), which was what we configured our SSI handler for. If we had ended the file with an `html` extension (`index.html`), our special codes would be heartlessly ignored. Speaking of special codes, our newly created file also contains our first introduction. Let's dissect what we see:

- This SSI statement and, in fact, all SSI statements, are encased in an HTML comment tag. If you accidentally include one in a file that is not handled by the SSI module, you can always "view HTML source" in your browser and see the statements unchanged. This becomes an important barometer: if your SSIs aren't working, then "view source" and see if they're being interpreted at all. If they are, they'll disappear from the final browser output – if they're not, you'll see them as normal HTML comments.
- After the opening comment tag, a # immediately appears. No spaces should exist between the two – if some do, then your SSI statement won't be interpreted correctly.
- We *virtually* include a file, `quote.shtml`, which doesn't yet exist. As specified, this file will be in the current directory (being `/Library/WebServer/Documents`). We can also use the standard `..` shortcut to point to files outside the current location.

Before we create our second file, load `http://127.0.0.1/index.shtml` in your browser. You'll be greeted by the error message in **Figure 1**. The cause of the error should be obvious: since `quote.shtml` doesn't exist, our first attempt at an SSI directive failed miserably.

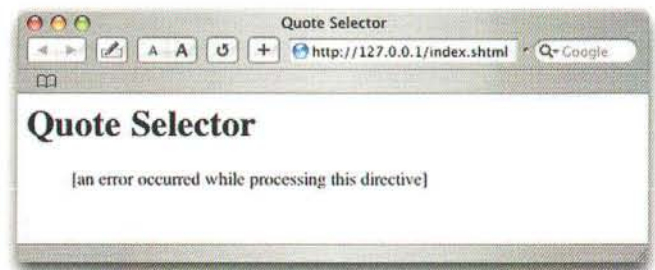


Figure 1: The not-so-pretty default SSI error message.

A rarely used feature of SSI, however, is the ability to customize this error message. Too many times have I visited sites and seen this error chuckling at the ineptitude of the web master. It's well-founded mirth, especially when the fix is mighty

mindless – as we can see below, we can customize the error message (with or without embedded HTML) for as many different uses as we need.

```
<html>
<head>
  <title>Quote Selector</title>
</head>
<body>

<h1>Quote Selector</h1>

<blockquote>
  <!--#config errmsg="Bah! Quote.shtml does not exist!"-->
  <!--#include virtual="quote.shtml" -->

  <!--#config errmsg="<br>Ouch! Nor does quote2.shtml!"-->
  <!--#include virtual="quote2.shtml" -->
</blockquote>

</body>
</html>
```

An example of this output is shown in **Figure 2**.

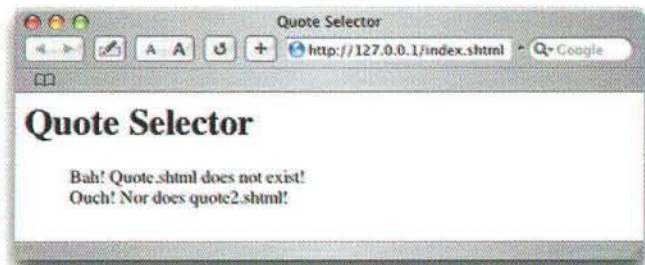


Figure 2: Customized SSI error messages for missing files.

Let's create our `quote.shtml`, saving into `/Library/WebServer/Documents`:

```
Toynbee Idea
<br>In Kubrick's 2001
<br>Resurrects Dead
<br>On Planet Jupiter
```

With the new file in place, reloading our browser shows us **Figure 3**. Note that we didn't actually need to name our data file with the `shtml` extension (like `quote.shtml`) – only the file that contains actual SSI statements need follow that restriction (so `quote.txt`, `quote.ssi` and `quote.include` would all have been viable alternatives). We'll be expanding `quote.shtml` with it's own SSI's shortly.

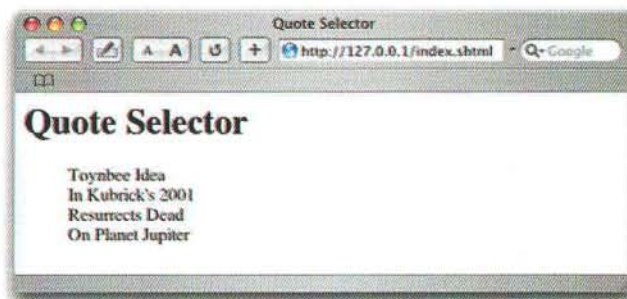
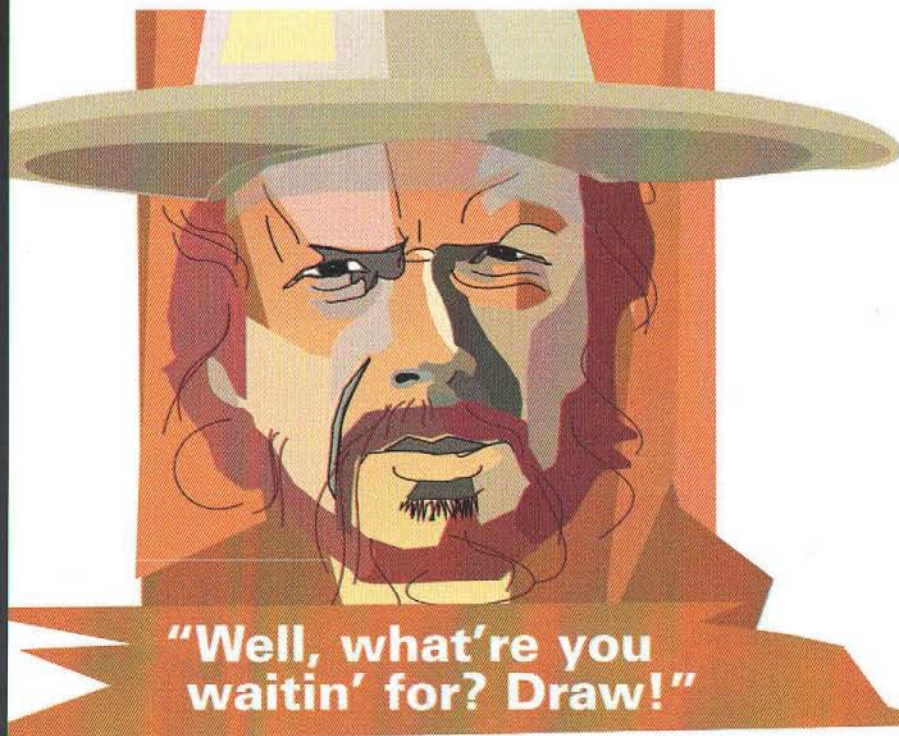


Figure 3: Success! Our first SSI is working as we intend.

Now, for literal purposes, this demonstration is winning no awards. The raw capability of SSI's, as we've mentioned, works best when combined with navigational elements,



Eazy Draw

Make Drawing Fun on OS X



Introducing EazyDraw – the fun, easy-to-use Mac OS X design tool that lets you draw like a pro! Now you don't need to be a graphic artist to create great illustrations. EazyDraw's vector-based graphics and editing capabilities make it easy to create technical diagrams, flow charts, and business communications as well as commercial line art illustrations and graphic elements for application software and web design.

Learn more about EazyDraw today! Get big savings buying direct from our online store. Visit us at **www.eazydraw.com** (That's easy with a Z).

© 2003 Dekorra Optics, LLC. All rights reserved. EazyDraw and the 'box of tools' are trademarks of Dekorra Optics, LLC. Mac and Built for OS X are trademarks of Apple Computer, Inc.



copyright statements, etc. Let's continue on with something a little more complicated.

PLAYING WITH SERVER SIDE INCLUDES: CONDITIONALS AND QUERIES

We've named our example "Quote Selector" for a reason: we'd like to offer a few different quotes that people can click on, but we don't want to have one page for each quote (like we would with `quote1.html`, `quote2.html`, `quote3.html`, etc.). We can do this easily enough with SSI conditionals and GET queries.

When you're submitting data through a web browser (as in typing your address into a form at Amazon, or choosing different result sets from a database listing), you're transmitting the data in one of two ways: GET or POST. GET's are for general-purpose forms, and are often used when you're simply requesting information: a search result from Google or a query match from a database. You can always tell when you've just used a GET form, because the resultant URL will contain the information you submitted. For instance, searching for "biozombie soda" creates a URL like `http://google.com/search?q=biozombie+soda`, where a value of `biozombie+soda` was assigned to a variable named `q`.

The prime benefit of GET is that you can bookmark the above URL and revisit it at a later date. POST's, on the other hand, are generally used when the site is requesting a lot of data (like a cut-and-paste of your high-school transcript). Unlike GET, the information submitted with POST is not transmitted in the URL, so it's not something readily recreated.

We're going to edit our two files so that they'll return different quotes depending on the contents of a GET query. Replace your existing `index.shtml` with the following, which we've added a selectable list of quotes to. Each quote is linked to the current document (since there's no filename specified), and passes a certain value through a GET query (either `q01` or `q02` – even though they don't have values, they'll be passed in our query string).

```
<html>
<head>
  <title>Quote Selector</title>
</head>
<body>

<h1>Quote Selector</h1>

<h2>Choose a quote:</h2>
<ul>
  <li><a href="?q01">On pavement.</a></li>
  <li><a href="?q02">On papyrus.</a></li>
</ul>

<blockquote>
  <!--include virtual="quote.shtml" -->
</blockquote>

</body>
</html>
```

And replace your existing `quote.shtml` with this next iteration, which contains a couple of noticeable additions, most prominent of which is a set of conditionals for testing against the value of `$QUERY_STRING`. Conveniently enough, the `$QUERY_STRING` is the entire value of the GET that would be

submitted from our newly revised `index.shtml`. If neither quote was chosen (ie. this was our reader's first visit to the page), then we spit out a quick warning that no quote has been chosen.

```
<!--if expr="\${QUERY_STRING}" = "q01" -->
  Toynebee Idea
  <br>In Kubrick's 2001
  <br>Resurrects Dead
  <br>On Planet Jupiter

<!--elif expr="\${QUERY_STRING}" = "q02" -->
  That is not dead
  <br>which can eternal lie
  <br>And with strange aeons
  <br>even death may die

<!--elif expr="\${QUERY_STRING}" = "" -->
  No quote has been selected!
<!--endif -->
```

Figure 4 shows the second quote having being selected, and the generated URL.

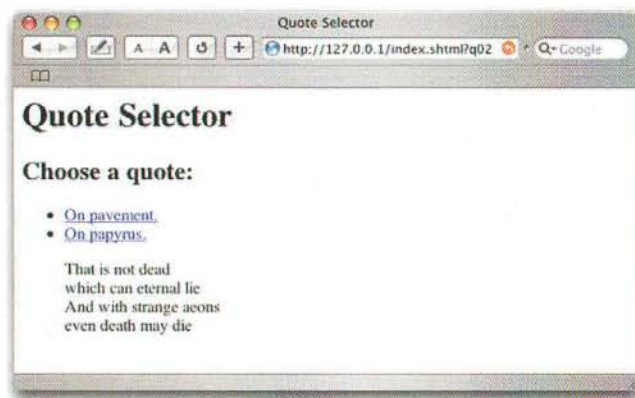


Figure 4: Our second quote displayed – notice the URL.

There's still more useful things you can do with this sort of structure: I've written before on using this technique to allow end users to customize positioning, specific stylesheets (which could radically change the layout, colors, fonts, etc.) and so forth, all without requiring the need of a cookie, and all choices being bookmarkable from computer to computer. Read more at "Allowing Simplistic User Preferences with SSI" (<http://hacks.oreilly.com/pub/h/226>).

I've also recently used this technique in my header files to change the logo that appears based on the specific URL (ie. `example.com/category1/` would have a different logo than `example.com/category2/`). To implement this, you'd use a few other tricks of SSI, namely the `$DOCUMENT_URI` variable, which contains the currently requested URL, as well as SSI's own getter (echo) and setter (set) methods. A quick example is below:

```
<!--if expr="$DOCUMENT_URI" = "/books_and_related/" -->
  <!--set var="header" value="header_books.gif"-->
<!--elif expr="$DOCUMENT_URI" = "/comics_and_zines/" -->
  <!--set var="header" value="header_comics.gif"-->
<!--else -->
  <!--set var="header" value="header_main.gif"-->
<!--endif -->
```

```
"
width="445" height="90" align="middle" hspace="5">
```

Seeing that I'm running out of space, it's best for me to link to some other SSI related hackery I've written, all of which demonstrate certain functionalities I've deigned to ignore here. "More Server Side Trickery" (<http://hacks.oreilly.com/pub/h/222>) covers cheap username and password authentication, different images or greetings depending on the current time, and server side hit counters and the powers of `exec`. A "Search Engine Friendly Image Gallery" (<http://hacks.oreilly.com/pub/h/225>), however, is a "full application", much like the quote selector above. It demonstrates how to use one SSI file to showcase an infinite number of images, with error correction, file size, modification times, and more.

FINISHING UP: WAYS TO MAKE THINGS BETTER

You may have noticed that we've been referring to `http://127.0.0.1/index.shtml` instead of the cleaner `http://127.0.0.1/`. Depending on whether you deleted the default web server files or not, you may have received an error message or directory listing when you requested the shorter URL. Why doesn't `index.shtml` display when we don't specifically request a document (as in `http://127.0.0.1/` or `http://127.0.0.1/testbed/`)? The answer, in a word: `DirectoryIndex`.

Apache's `DirectoryIndex` controls which files it should consider the default document for a directory – in other words, what should be served when no other file has been requested. By default, this is normally just `index.html`, but you can include as many fallbacks as you wish, as per the following example:

```
DirectoryIndex index.shtml index.html index.cgi default.htm
```

When you're editing this line in `/etc/httpd/httpd.conf`, be sure to list the file names in order of preference and usage: if you're going to be using `default.htm` more often than `index.cgi`, move that to earlier in the list. You'll get small performance gains by sorting correctly like this as Apache won't have to look through the entire list for each request.

HOMEWORK MALIGNMENTS

In our next column, we'll chat about CGI: what it is, how to enable it, how to code for it, how to implement scripts you find on the 'net, and all the rigmarole and hilarity that ensues. If we have room, we'll also talk about how to remove the need for file extensions, definitely an important step to good URL design (see our first column). For now, students may contact the teacher at morbus@disobey.com.

- "I pity any" *what* "who isn't me tonight"?
- What other inventive things can SSI be used for?
- The quotes in our examples: where'd they come from?
- Ever seen *Biozombie*? Any similar suggestions?

There are ^{data}Users and ^{data}Losers...

Which
would you rather
be

Introducing
Data Safety System
Backup, Undelete and Recover files.



Data Users get it!

www.prosofteng.com
PROSOFT
engineering, inc.

These products are only available for the Mac OS, so your
Windows friends will still be losers...

©2003 Prosoft Engineering, Inc., All rights reserved



By David Linker

Scripting Nisus Writer Express

Writing perl macros to add functionality

NEW OPPORTUNITIES WITH MAC OS X

Although we have all read about how the change from OS 9 to OS X has challenged developers as they adapt their programs to either Carbon or Cocoa, the addition of BSD has added a number of powerful tools that can be accessed from or added to applications. An example of this is the powerful word-processing program Nisus Writer, which is especially well suited for writing in foreign languages, and has a very powerful macro programming language. The current version runs in Classic, but the programmers are hard at work on a Cocoa version, and have released a new version, which they call Nisus Writer Express. This version is available for evaluation as a free download, at [<http://www.nisus.com/Express/>](http://www.nisus.com/Express/).

A PERL BY ANY OTHER NAME

One of the interesting design decisions that the programmers made was that they incorporated the ability to run perl scripts as a macro language. Perl is an interpreted language that is now included on every Macintosh as part of OS X. The name "perl" comes from "Practical Extraction and Report Language", and in the words of the documentation "Perl is a language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information." It has been around since 1987, and is a favorite of many for processing text files. It has a rich set of text processing capabilities, many useful libraries, and an active community of users. As such, it is an excellent choice for a macro language.

In order to try out the capabilities of this new type of macro extensibility, and learn a little about perl in the process, I decided to add back a simplified version of a feature of the Classic version of Nisus that is currently lacking in Nisus Writer Express, namely the mail merge function.

In the process of implementing this functionality, we can learn about how the programmers of Nisus Writer have connected perl and Nisus, and also learn a little about perl programming.

MAKING A MACRO

Assuming you have installed the Nisus Writer Express beta, we can start making a perl macro. Start up Nisus Writer Express, which we will call NWE from now on. After it has started up, select "New Perl Script" from the Macro menu. In the window that comes up, type in the following:

```
#!/usr/bin/perl

#Nisus Macro Block
#source front
#destination new
#End Nisus Macro Block

use strict;
use warnings;

print 'Mail merge coming soon!';
```

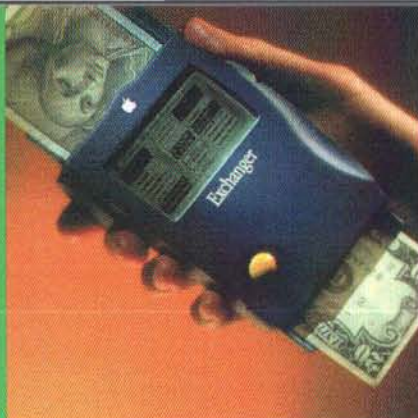
Save this by choosing "Save as Macro..." from the "Macro" menu. In the dialog that comes up, you will need to choose "Nisus Perl Macro" from the file format pop-up, and the "UTF-8" choice from the text encoding. You should choose the name, which should be "Mail Merge.pl". Note that the extension should be filled in automatically if you have chosen the perl macro format. Check that the destination for the file is the /Users/<user_home>/Library/Application Support/Nisus Writer Express/Macros, where "<user_home>" represents where your user directory is, and save the file.

Now, we will try running the file before we dissect it. Choose "Mail Merge" from the macro menu. A new window should open up, and the line "Mail merge coming soon!" should be written in it.

Now, we will look at how this works. The first point to understand is that any line that starts with a pound sign is considered a comment, and is ignored by the perl interpreter. This means that the first line is a comment, but this is interpreted by the shell as instructions on where to find the interpreter for the script, which in this case is perl. In the case of a Nisus perl macro, this is not actually necessary, since NWE knows that a macro with the ".pl" extension is to be executed as perl, but it is good form to include it since all other perl scripts outside of NWE will need this line to function properly.

David is a collector of computer languages, having programmed in Algol-60, Fortran IV, Pascal, Basic, Lisp, Forth, Logo, Modula-2, java, and various flavors of assembly. He has also written some C, but tries not to admit it. He is currently teaching himself Objective-C, and scripting under Mac OS X. You can reach him at dtlinker@mac.com.

32
< 1 year
2 years >
\$64



INTERVIEWS

TAPPING INTO THE WORLD OF CELEBRITIES AND THEIR MACS, ONLY MacDIRECTORY OFFERS EXCLUSIVE INTERVIEWS. GET A CLOSE AND PERSONAL VIEW FROM SARAH JESSICA PARKER, STING, STEVE JOBS, MADONNA, HARRY CONNICK JR., GEORGE LUCAS, JENNIFER JASON LEIGH, STEVE WOZ AND OTHER LEADERS IN THE MAC COMMUNITY.



FEATURES

DESIGNERS, WRITERS, MUSICIANS, BUSINESS LEADERS & OUR TECHNICAL EXPERT TEAM OFFER THEIR OWN PERSONAL INTERPRETATION OF THINGS THAT ONLY THE MAC SYSTEM CAN DELIVER. FEATURING OVER 240 PAGES OF REVIEWS, INTERVIEWS, NEWS, INSIGHTS, TRENDS AND THE LARGEST MACINTOSH BUYERS' GUIDE INCLUDING OVER 5,000 MAC PRODUCTS AND SERVICES.



CULTURE

MacDIRECTORY TAKES YOU TO THE WILDEST CORNERS OF THE WORLD AND UNCOVERS HOW MACINTOSH COMPUTERS ARE BEING USED BY OTHER CULTURES. TRAVEL TO JAPAN, AUSTRALIA, GERMANY, BRAZIL, INDIA, RUSSIA & LEARN MORE ABOUT APPLE'S CULTURAL IMPACT AROUND THE GLOBE.

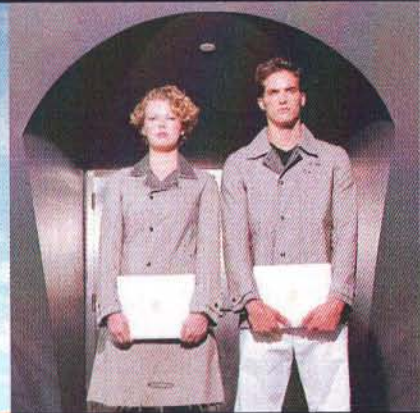
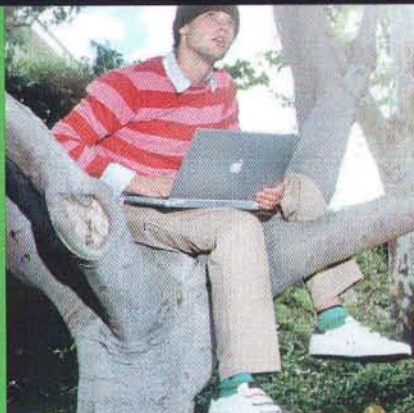
MacDirectory

BEYOND ANY MACINTOSH MAGAZINE. SUBSCRIBE.

< Subscribe

www.macdirectory.com/mw.html

SEND CHECK OR MONEY ORDER TO:
MacDIRECTORY SUB DEPT.
326 A STREET, 2C
SOUTH BOSTON, MA 02110



REVIEWS

FIND OUT ALL YOU NEED TO KNOW ABOUT THE LATEST MAC PRODUCTS INCLUDING THE HOTTEST MAC OS SOFTWARE AND HARDWARE.



WIN!

SUBSCRIBE TO MacDIRECTORY AND YOU WILL AUTOMATICALLY ENTER OUR SWEEPSTAKES FOR A CHANCE TO WIN A NEW TITANIUM!

The next four lines are special comments that communicate with NWE to help set up the environment in which the macro runs. These take the form of a number of optional commands. The first line, "**Nisus Macro Block**", defines the beginning of these instructions. The command "**source**" defines where standard input will come from. The options are to take it from the front window (**front**), the window below the front window (**next**), the clipboard (**clipboard**), or no input (**none**). The next line is the command "**destination**", which defines where standard output will go, which includes the same options as the "**source**" command, with one addition, which opens a new window for output, with the "new" option. If no other option is specified, the output is appended to the end of the current contents. If you want to replace the current contents, you add the "**replace**" option. The "**End Nisus Macro Block**", ends the Nisus commands section.

The next two lines are commands to the perl compiler, which control how it handles ambiguous situations. The first, "use strict", invokes some error checking, such as making sure that variables are declared before they are used. This can be very helpful, since if you don't use this option and you type a variable name incorrectly, it will just create a new variable, introducing a hard to find error. The second reports warnings, which include use of uninitialized variables, dropped variables, using closed or undefined file handles, among others. These commands are not necessary, but are often recommended to learners of perl, like myself, in order to enforce good programming practices.

Finally we get to the action statement, which is a simple print statement that prints a string.

MERGING MAIL

We are now ready to write our first version of the mail merge function. We will try to duplicate the original mail merge function of Nisus Writer. There were two files that had to be created. There was a template document and the merge file. The template document had all of the common text, with the variable text replaced by keywords surrounded by "international quote marks", or " and ", also known as a left and right guillemot (which is also the name of a bird. Go figure!). You can type the " " by pressing the option key, and then the backslash, or "<option>-\". The " " can be typed by pressing "<option>-<shift>-\". The merge document in its simplest form consists of comma-delimited text, with the first line containing the keywords, and subsequent lines the text to be substituted for each keyword, with each line representing a different copy of the template.

With this in mind, we will create a new version of the macro, which will imbed the template in the macro itself, and read the merge file from the front window. Reading in a template is more complex, and we will deal with that later. The output will be put in a new window. With that in mind, type in the following and save it.

```
#!/usr/bin/perl

#Nisus Macro Block
#source front
#destination new
#End Nisus Macro Block

use strict;
use warnings;

my @template = ("Dear <firstname>.\n",
"Thank you for the <gift> you gave me.\n");

my @mergefile = <STDIN>;

my $line = ''; # Holder for temp strings

foreach $line (@mergefile) {
    chomp ($line);
}

my @keywords = split(/,/, $mergefile[0]);

foreach $line (@keywords) {
    $line = '<'. $line. '>';
}

for (my $i=1; $i<@mergefile;$i++) {
    my @copy = @template;
    my @subst = split(/,/, $mergefile[$i]);
    for (my $j = 0; $j < @subst;$j++) {
        for (my $k = 0; $k<@copy; $k++) {
            $copy[$k] =~ s/$keywords[$j]/$subst[$j]/g;
        }
    }
    print @copy;
    print "\n\n";
}
```

Now we need to create the merge document. Open a new window, and type the following:

```
firstname.gift
Joe, stuffed moose
Bob, old barrel
```


If you want, you can save this file as a text file.

Now, with that file as the foreground file, run your macro from the macro menu. You should get the following output:

```
Dear Joe,
Thank you for the stuffed moose you gave me.

Dear Bob,
Thank you for the old barrel you gave me.
```

Let's see how this works. The beginning and set-up are the same as before. We then create an array of strings that holds the template. The "my" keyword defines the following variable to be local to the enclosing block. A variable that starts with a "\$" is a scalar, or single variable, which can hold a number or string. A variable that starts with a "@", on the other hand, is an array, which can also hold numbers or strings. In this case, we are declaring a local array variable named @template, and assigning two strings to this array. Note that the strings are surrounded by double quotes rather than the single quotes we used in our first version. If we use single quotes, the strings are used exactly as they are typed. When we use double quotes, the enclosed string is interpreted by a process called



What does software protection
have to do with money?

11 billion US\$ lost worldwide
through software piracy!

WIBU-KEY Software Protection

■ Software goes online

Electronic Software Distribution –
safely protected by WIBU-KEY.

■ Pay-Per-Use

Usage dependent accounting of
your software.

■ License Management

You can easily and flexibly create and
manage network licenses.

■ Mac OS 9 & X

WIBU-KEY supports Mac OS, Windows
and heterogeneous networks.

Test the WIBU-KEY Protection Kit
free and decide for yourself!

1-800-986-6578

sales@griftech.com



The Key is in Your Hands!

WIBU
SYSTEMS

WIBU-SYSTEMS USA, Inc.
Seattle, WA 98101
email: info@wibu.com

www.griftech.com
www.wibu.com

Test Kits also available at:

Belgium wibu@impakt.be, Denmark lean@danbit.dk, Finland finbyte@finebyte.com, France info@neol.fr, Hungary info@mrsoft.hu, Japan info@suncarla.co.jp,
Jordan, Lebanon starsoft@cyberia.net.lb, Korea dhkimm@wibu.co.kr, Luxembourg wibu@impakt.be, Netherlands wibu@impakt.be,
Portugal dubit@dubit.pt, Thailand preecha@dpf-th.com, United Kingdom info@codework.com, USA sales@griftech.com

"interpolation" in the perl documents. Any included variable names are evaluated, and the result placed in the string. Also, special sequences are replaced. We use this characteristic using the "\n" combination to indicate to insert a carriage return after each line.

Next, we create another local array variable, and use it to read from STDIN, the standard input, which has been set to read from the front window. We use a trick in perl, which is that we can read an entire file with one statement into an array, and it will be automatically split into lines. Each line will still have the carriage return at the end, which can mess us up if we don't remove it. We do this in the `foreach` loop, using the `chomp` function, which removes the last character of a string. The `foreach` function is like a `for` loop but without explicitly indexing every element in the array. In this case, the variable `$line` is assigned to each of the elements in the array `@mergefile` in order, and changing `$line` will result in changing the array element. This frees us from the indexing details.

We then need to find the keywords from the first line of `mergefile` array. To do this, we use the `split` function. It takes two arguments. The first consists of a delimiter, followed by what sequence of characters signifies a split point, followed by the delimiter again. In our case, the comma is signified as our split character. The second argument is a string which is to be split. The function returns an array containing the fragments that resulted from the split, in this case, the keywords. We then add the international quotes to each of the keywords, so that we will find the keywords only when they are contained within the international quotes.

We are then ready to process the merge file data. This is done with a set of nested `for` loops. Note that if we use an array identifier in a context which calls for a single (scalar) variable, it returns the number of elements in the array. In order to access array elements, we use the array name preceded by a "\$", and followed by square brackets enclosing the index, which starts with zero. The first `for` loop is repeated with each line of the merge data, and creates the array of strings that will replace the keywords. In it, we make a copy of the template to work on, and split the string with the merge data in it. We next have loops for each substitution, and for each line in the template. In the inside of the loop, we perform the actual substitution, using the "s" command. The `=~` means "operate on the variable to the left, and then replace it with the result on the right". The portion between the first two slashes is the string to match, while the second portion is what will replace it. Each of these portions is treated as if it were a string in double quotes, meaning that it is interpolated. This means that variables are replaced with their values. The "g" at the end means to repeat the substitution as many times as possible.

ADDING FLEXIBILITY AND STYLE

The version we have lacks flexibility, because to change the template text we have to change the macro. It would be

better if the macro read the template text from a file. A perl macro in Nisus can read data from a file, from the front window, from the next window, or the clipboard. We could put the template in one of these locations, but we would need to be certain that either the clipboard contains the correct data, or that we have the correct windows open and in the correct order. In order to keep things simpler and more reliable, we will instead use a file for the template in combination with the front window as the merge file.

A second deficiency is the lack of the ability to use different fonts and styles. Perl is designed for plain text, and when it is reading from the window within Nisus it only extracts the text, without any font or style information. One way we can convert all of the style and font information to plain text is to save the file as RTF, which is a text representation including all the page setup, font and style information. If we do this, we can turn our previous decision to save the template as a text file to our advantage.

A deficiency of using perl this way is that we can't have user input to a running perl script. We would like to be able to specify where the merge template file is, but if we include it in the merge data file, it would be different than the syntax of the previous version of Nisus uses for its mail merge function, which we are trying to emulate. Instead, we will use a built-in feature of the NWE implementation of perl to find the merge template in the same location as the merge file.

The NWE allows us to get the full path to the file associated with the frontmost window in the second element of a special array, as `$ARGV[1]`. This array is normally used in perl to access command line options, and if there is no front window, or the front window has not been saved, the array will be empty. If I have a file named "test.rtf" in the Documents folder of my home directory (dtlinker), the path will be `/Users/dtlinker/Documents/test.rtf`. We use this string to construct the file name and path for the template and output files.

So, enter the following in Nisus Writer Express, and save to your Documents folder as "Merge template.rtf", in RTF format. Note that the keyword for the firstname is in bold, and the keyword for the gift is in italic and in a larger font size, to demonstrate font style changes. When you change these, be careful to include the international quotes on either side when you change the size or style.

```
Dear <firstname>,  
Thank you for the <gift> you gave me.
```

We also need to decide where to send the output. The output will be an RTF file, which means that if we just send it to a window, we will get all of the unintelligible format specifications. We could do this, and then save the file with an ".rtf" suffix, and then open it again to see the changes, but this would add extra steps. Theoretically, we could alternatively use the Nisus perl macro directive `"#Send Text as RTF"`, and get the formatting interpreted correctly in the window, but when I tried

this I ran into erratic behavior. So, we will save the output directly to a file with the ".rtf" suffix, to avoid these problems. This will also avoid the extra step of saving and reopening the window with the results.

So, the plan is to read the template from an RTF file, process it using the merge file which is located in the front window, and then save the output to another RTF file. There is just one other change that we need to make. When we save a keyword of the format "keyword" as RTF, the international quotes are altered to the special codes, "\'c7" and "\'c8". This means that what we are going to look for is a string "\'c7keyword\'c8".

Now, things start to get complicated. The character "\", or backslash is special in perl, and is the escape character, which means that the next character is to be interpreted in a special way. So, in order to get a backslash, we need to use two in a row. Now, recall that the string in the substitution command `s` is interpolated. This means that we need two backslashes in a row to get one in the search string. Since the search string is interpolated, and double quoted strings are interpolated, in order to get a single backslash we need to have four backslashes in a row!

BELLS AND WHISTLES

There are a few more final details we need to take care of. The first is error detection and handling. The main errors that can occur are when we open a file, or when we try to find the path associated with the front window. We can trap these errors by using the return value of the open function, and the `die` function. This will send the subsequent message to `STDERR`, or the standard error output, and then exit the program. For perl embedded in `new`, `STDERR` is output to a modal dialog.

Another trick that adds a nice feature is the ability send a command to the shell from within a perl script. This is done by enclosing the command with backwards slanted single quotes, located at the upper left of the keyboard on my iBook. The last line of this macro opens the resulting output file in `NWE` using this technique.

The final version is listed below.

```
#!/usr/bin/perl
# Mail Merge.pl
# A mail merge macro for Nisus Writer Express
# written by David T. Linker
# 7/1/03 Version 1.1
# dtlinker@mac.com
#
# Uses template and merge file format of Nisus Classic
#
# Save the this file in the macro folder in the folder
# <user home>/Library/Application Support/Nisus Writer/Macros
#
# Make a template file, with the keywords surrounded by
# guillemets, like this: «keyword». You can get these characters
# by using <option>-\ and <option>-<shift>-\. Save the template
# file as RTF with the last part of the name before the extension
# being " template".
#
# Make a merge data file, with the first line consisting of the
# keywords separated by commas, then the subsequent lines with
```

New

PrimeBase 4.2 Replication Server

Check out the fully programmable Replication Server

- Bidirectional Updates supported
- Update 3rd-party DBMS
- Send Emails
- Post/get Data to/from Websites

SQL-Runtime Plugin
for REALbasic
\$ 499,-

All PrimeBase Server Software

- SQL-Runtime Libraries available
- SQL Database Server
- Application Server
- Replication Server
- Open Server

Available on the most popular platforms

- Completely cross-platform
- Full-text searching and indexing
- Mac OS & OS X
- Linux
- Solaris
- IBM AIX
- all Windows platforms

 **PRIMEBASE**

SNAP Innovation GmbH
Altonaer Poststraße 9a
D-22767 Hamburg / Germany
www.primebase.com
e-mail: info@primebase.com
Fon: ++49 (40) 389 044-0
Fax: ++49 (40) 389 044-44

```
# the text to substitute separated by commas. Save the file to the
# same folder as the template file, with the same name as the template
# file, but without the word "template".
#
# Leave this file open as the front window, and run the macro.
# The output will be opened in a new window, and saved to the
# same directory as the other files.
#
# Known bugs:
# Ill-formed RTF output
# Does not handle accented characters in merge data
#
# Valuable suggestions and additions by Kino, of the Nisus list

#Nisus Macro Block
#source front
#End Nisus Macro Block

use strict;
use warnings;

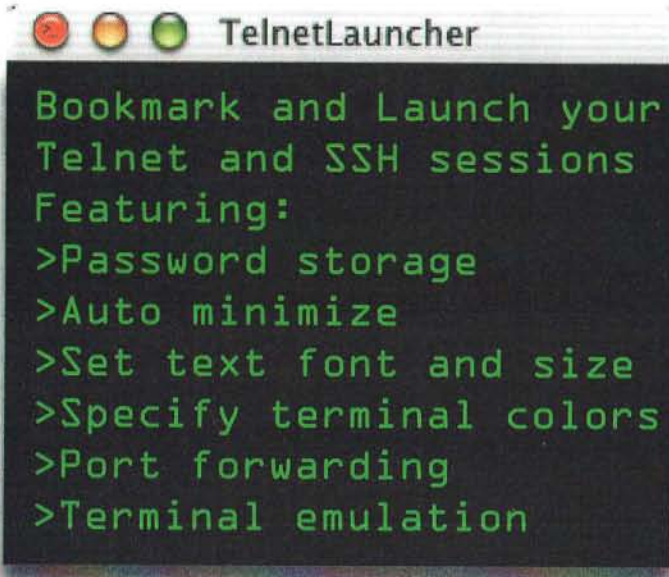
# Is there a front window file, and has it been saved?
if (@ARGV<1) { # No arguments means no merge file
    die "Front window not saved, or no front window\n";
}

# Now, make the name of the input and output files
my $merge_template_file = $ARGV[1]; # Gct file path

$merge_template_file =~ s/\.([0.3])$//; # Remove extension
my $merge_output_file = $merge_template_file.' OUTPUT.rtf';
$merge_template_file = $merge_template_file.' template.rtf';

# Read from file at root
open (TEMPLATE, '<', $merge_template_file)
    or die "Unable to open $merge_template_file";

my @template = <TEMPLATE>;
close(TEMPLATE);
```



This and much more available from...

piDog Software

SimpleKeys - piPop - DockSwap - ScreenShot Plus

www.pidog.com

Info@pidog.com

```
# Read merge data from front window
my @mergefile = <STDIN>;

foreach my $line (@mergefile) {
    chomp ($line);
}

# Extract the keywords
my @keywords = split(/./, $mergefile[0]);
foreach my $line (@keywords) {
    $line= "\\\\'c7". $line. "\\\\'c8";
}

# Open the output file
open (OUTFILE, '>', $merge_output_file)
    or die "Unable to create $merge_output_file";

#Do the actual merge
for (my $i=1; $i<@mergefile;$i++) {
    my @copy = @template; # make a local copy to alter
    my @subst = split(/./, $mergefile[$i]);
    for (my $j = 0; $j < @subst;$j++) {
        for (my $k = 0; $k<@copy; $k++) {
            $copy[$k] =~ s/$keywords[$j]/$subst[$j]/g;
        }
    }
    print OUTFILE @copy; # Output the result
    if ($i<@mergefile-1) {
        print OUTFILE '\page'; # Insert a page break between copies
    }
}

close (OUTFILE);

# Let's look at the result
`open -a "Nisus Writer Express" "$merge_output_file";
```

To use this macro, you should make the merge data document and save it, leaving it as the front window. If it has the name "Merge", with any kind of extension, make sure that you have your template stored as "Merge template.rtf" in the same folder as the merge data document. If you then run the macro and it runs successfully, there will be no messages, but you will have a file named "Mail Merge OUTPUT.rtf" in the same folder as the other files, and open as the front window.

FUTURE IMPROVEMENTS

This version accomplishes all of our original goals, which were to implement the basic functionality of mail merge, and learn some perl in the process. There are a number of improvements that could be added, and deficiencies addressed. Some of these are:

- We could add error checking for input data formats of the merge template and merge data.
- The output is not syntactically correct RTF, which leads to some format errors when the result is opened in other applications. We could fix the RTF by actually parsing the RTF and only changing the text, and adding page breaks between copies.
- Accented text in the merge data is not handled properly, since extended ASCII, which is read from the merge data file, is handled by special codes in RTF, which is the format for the template and output files. Adapting the macro to use RTF for the merge data file as well would fix this.

- We could implement other features of the mail merge language, which included conditionals and include files, for example.

RESOURCES

There are a number of resources that I found useful in working on this, and which may prove useful to those who wish to write perl macros.

There were two tutorials that I found to be very helpful. The most basic, which I used most extensively, was at the University of Leeds site, at:

[<http://www.comp.leeds.ac.uk/Perl/start.html>](http://www.comp.leeds.ac.uk/Perl/start.html)

A more detailed tutorial that I also used was found at:

[<http://www.ebb.org/PickingUpPerl/pickingUpPerl_toc.html>](http://www.ebb.org/PickingUpPerl/pickingUpPerl_toc.html)

Full documentation of perl is available on your Macintosh by using the Terminal and typing "man perl" on the command line. This accesses the main documentation guide, which also has links to the other documentation. A very useful online resource for the perl manual pages, documentation and function descriptions, among other things, is found at:

[<http://www.perldoc.com/>](http://www.perldoc.com/)

One of the documentation pages that may be of interest is "perlembed", which describes "how to embed perl in your C program". In addition to the documentation, this site also contains links to many other sites of interest to perl

programmers. One that I found quite intriguing is CPAN, or "Comprehensive Perl Archive Network":

[<http://www.cpan.org/>](http://www.cpan.org/)

This contains a vast library of library routines for perl. One set that I looked at for this article was the modules to parse RTF files, but these would require independent installation to be used as is, and I felt that this would be more complex than I wanted for this demonstration. There are even routines for accessing URL's over the internet!

The Nisus Writer Express perl macro usage is documented in the help file, accessed by the Help menu, then clicking on "Macro Menu", then on "More about Nisus Writer Express Macros". Additional information is included in the release notes for Nisus Writer Express, which are accessible from the download page at Nisus.

Finally, an extremely valuable resource for those who use Nisus Writer is the Dartmouth Nisus list:

[<http://listserv.dartmouth.edu/archives/nisus.html>](http://listserv.dartmouth.edu/archives/nisus.html)

Members on this list are very helpful and knowledgeable about Nisus and a wide range of other topics. One of the list members, Kino, was particularly helpful with suggestions and techniques that were used in this macro, including the error checking on file opening, and the technique of executing a shell command to open the result file in a Nisus window.

I hope that this will spur your interest in perl in general and writing perl macros for Nisus Writer Express in particular. Maybe you will even be inspired to embed perl functionality into an application you are developing. Have fun!



Now serving Cocoa[®] just the way you want it.

Training for Mac OS X doesn't have to be the same old flavor. Reserve your seat in a class at our scenic lodge location, or have experts come to you for **Extreme Mentoring**. Two weeks of on-site instruction and collaboration, customized to the requirements of your project. Book now for 2003. See why we're different.



**Big
nerd
ranch**

Intensive Classes for Programmers
www.bignerdranch.com

By Richard Patterson, Los Angeles, CA

X Files Carbonara

Making Navigation Easier for the Impatient

THE GOOD OL' DAYS

Before I was coerced into being carbonized, I had a simple scrap of code I could grab and use whenever I needed my application to create and write a file.

```
long      byteCounter;
StandardFileReply reply;
FSSpec    asciFile;
short     asciFileNum;
char      *textData;
OSError   err = noErr;

StandardPutFile("\pSave Text Data as:",
"\pFullLookupTable.txt", &reply);
if(reply.sfGood)
{
    asciFile = reply.sfFile;
    FSpDelete(&asciFile);
// ignore any error caused if there is no such file
    err = FSpCreate(&asciFile, 'XCEL', 'TEXT', -1); /* -1 =
system script */
    err = FSpOpenDF(&asciFile, fsCurPerm, &asciFileNum);
    if (err != noErr) return err;

    err = FSpWrite(asciFileNum, &byteCounter, textData);
    if (err != noErr) return err;

    err = FSpClose(asciFileNum);
}
return err;
```

It couldn't get much more straightforward than that. Those were the days when the Toolbox really made life simpler for a programmer. It helped me tell the user (generally myself) what he was supposed to do and even suggest a default name for the file. The only suspect part of this code was the shortcut method of insuring that the file was indeed a virgin file by attempting to delete it before (re)creating it. I knew there were more elegant ways to let StandardPutFile tell me that the user was replacing an existing file rather than just creating a brand new one, but most of the time I didn't care. I just wanted to write the file and get on with it. *(Most of my programming is for in-house use only, so I can get away with quick-and-dirty solutions that you should only try at home.)*

Recently I was desperately trying to debug an After Effects plug-in to send images to a film recorder, and I realized the only

way I was going to be able to tell what was going on would be to save a file capturing the state of the image at a certain point. I don't write applications that write files all that often, but years ago I acquired the prejudice that reading and writing files is one of the most basic functions the operating system needs to do and should therefore be a very simple programming task. So I resurrected a scrap of code designed to save an image buffer as a simple Photoshop file and threw it into the soup.

The next thing I knew I was spending two days trying to figure out how to replace the method shown above with something that would work inside my carbon code running on OS-X. I found myself floating around in all kinds of convoluted discussions of Apple Events and Unicode text. I eventually vented my frustration on Apple Developer Support whose suggestions for further reading and examples only seemed to complicate what I thought I was beginning to grasp. Fortunately the support technician was very patient, and I was able to crystallize my ferment into a rational suggestion that OS-X should provide a much simpler higher level function to help the less experienced programmer create and write a file. The support technician agreed that was a good idea, but indicated that it was not at the top of their priorities. When I finally saw the light thanks to Mr. K.J. Bricknell's indispensable tome, *Carbon Programming*; I realized that perhaps I should write a sample function that might spare someone else the agony I had just experienced.

NON STANDARD

The functions for opening and writing to a file (FSpOpenDF and FSpWrite) still work in Carbon on OS-X, but the StandardPutFile and StandardGetFile functions have been replaced by Navigation Services. The system had just outgrown the functionality provided by the Standard File Package. There is a Navigation Services function NavPutFile that was the original replacement for StandardPutFile, but with OS-X Apple recommends that we use NavCreatePutFileDialog. If I had been keeping up, the transition from StandardPutFile to NavPutFile to NavCreatePutFileDialog might have been smooth and effortless. Instead I woke up and found the following definition staring me in the face:

```
OSStatus NavCreatePutFileDialog (
    const NavDialogCreationOptions * inOptions,
    OSType inFileType,
    OSType inFileCreator,
    NavEventUPP inEventProc,
```

Richard Patterson is in charge of digital imaging at Illusion Arts, a visual effects facility in Van Nuys, CA, specializing in matte paintings and bluescreen compositing for movies. You can reach him at richard@illusion-arts.com.

Ask yourself this question...

I use my Mac for:

- ☐ Programming as a hobby
- ☐ Exploring the depths of Mac OS X
- ☐ QuickTime Development
- ☐ Application Development
- ☐ Network Administration
- ☐ All Of The Above

...this is the answer:

 **MacTech[®]**

The Journal of Macintosh Technology and Development

Whether you program as a hobby or are a full-time developer, MacTech gives you the info you need from the people that know.

www.mactech.com

PO Box 5200 • Westlake Village, CA • 91359-5200 • orders@mactech.com
Toll-Free: 877-MACTECH • Outside US/Canada: 805-494-9797 • Fax: 805-494-9798

```
void * inClientData,
NavDialogRef * outDialog
);
```

Then I discovered there were 11 other functions I must call before I can have an FSSpec to use in the familiar way.

What I wanted was one function that took care of all the user interaction and just gave me a ready-to-wear FSSpec. It would need to know what kind of file I am trying to create, so there are three things I need to give it: the file type, the file creator and a pointer to the FSSpec.

```
OSError SimpleNavPutFile( OSType fileType,
OSType fileCreator,
FSSpec *theFileSpec)
{
    OSStatus theStatus;
    NavDialogRef theDialog;
    NavReplyRecord theReply;
    AEDesc aeDesc;
    FSRef fsRefParent, fsRefDelete;
    UniChar *nameBuffer;
    UniCharCount nameLength;
    FInfo fileInfo;
    OSError err = noErr;

    theStatus = NavCreatePutFileDialog(NULL, fileType,
fileCreator,
NULL, NULL,
&theDialog);
    NavDialogRun(theDialog);
    theStatus = NavDialogGetReply ( theDialog, &theReply);
    NavDialogDispose(theDialog);

    if(!theReply.validRecord)
    {
        // Assuming the user changed his/her mind? No harm; no foul.
        // Still need to indicate that a file has not been created
        return -1;
    }

    err = AECoeerceDesc(&theReply.selection, typeFSRef,
&aeDesc);
    if(err != noErr) return err;
    err = AEGetDescData(&aeDesc, &fsRefParent, sizeof(FSRef));
    if(err != noErr) return err;
    nameLength =
(UniCharCount)CFStringGetLength(theReply.saveFileName);
    nameBuffer = (UniChar *) NewPtr((long)nameLength);
    CFStringGetCharacters(theReply.saveFileName,
CFRangeMake(0, (long)nameLength),
&nameBuffer[0]);
    if(nameBuffer == NULL) return -1; // generic error
    if(theReply.replacing)
    {
        err = FSMakeFSRefUnicode(&fsRefParent,
nameLength, nameBuffer,
kTextEncodingUnicodeDefault,
&fsRefDelete);
        if(err == noErr) err = FSDeleteObject(&fsRefDelete);
        if(err == FBSyErr)
        {
            DisposePtr((Ptr)nameBuffer);
            return err;
        }
    }

    err = FSCreateFileUnicode(&fsRefParent, nameLength,
nameBuffer,
kFSCatInfoNone, NULL, NULL,
theFileSpec);

    err = FSPGetFInfo(theFileSpec, &fileInfo);
    fileInfo.fdtype = fileType;
    fileInfo.fdcCreator = fileCreator;
    err = FSPSetFInfo(theFileSpec, &fileInfo);

    return err;
}
```

So now my original scrap of code would become:

```
long byteCounter;
StandardFileReply reply;
FSSpec asciiFile;
short asciiFileNum;
char *textData;
OSError err = noErr;

err = SimpleNavPutFile('TEXT', 'XCEL' &asciiFile);
if(err == noErr) //a file was created
{
    err = FSpOpenDF(&asciiFile, fsCurPerm, &asciiFileNum);
    if (err != noErr) return err;

    err = FSWrite(asciiFileNum, &byteCounter, textData);
    if (err != noErr) return err;

    err = FSClose(asciiFileNum);
}
return err;
```

I've sacrificed a little functionality in the dialog, since I can no longer suggest a default file name and prompt the forgetful user about what he is supposed to be doing. This scrap is fewer lines of code than my original, though, and even easier to use. I shall not attempt to explain what all the functions are doing in my SimpleNavPutFile. All I can say is that this works on my machine and is not meant to be anything other than a quick and dirty solution. Note that it includes the call creating the file and deals with the choice to replace an existing file. It may just amount to the same thing as using the now-deprecated NavPutFile, but I believe it lets OS-X put up the latest and greatest file navigation dialog.

I should confess that this solution will fail with OS-9 because it gives up if it cannot coerce the AEDesc to an FSRef. Bricknell's book has a discussion on page 960 of how to derive the FSSpec in OS-9 when this coercion fails. I have not included it, because my immediate concern was getting over the hump in OS-X, and I want to keep this as simple as possible.

The corresponding SimpleNavGetFile is built around

```
OSStatus NavCreateGetFileDialog (
const NavDialogCreationOptions * inOptions,
NavTypeListHandle inTypeList, // can be NULL
NavEventUPP inEventProc, // can be NULL
NavPreviewUPP inPreviewProc, // can be NULL
NavObjectFilterUPP inFilterProc, // can be NULL
void * inClientData, // can be NULL
NavDialogRef * outDialog
);
```

Most of the parameters which can be set to NULL have system defaults that will be used when they are NULL. Setting the NavTypeListHandle to NULL simply results in no file filtering in the dialog. In order to avoid dealing with the NavDialogCreationOptions you can use NavGetDefaultDialogCreationOptions to set everything to a default.

```
SimpleNavGetFile(FSSpec *theFileSpec)
{
    OSStatus theStatus;
    NavDialogRef theDialog;
    NavReplyRecord theReply;
    NavDialogCreationOptions inOptions;
    AEKeyword theKeyword;
    DescType actualType;
    Size actualSize;
    OSError err = noErr;

    NavGetDefaultDialogCreationOptions(&inOptions);
```

```

theStatus = NavCreateGetFileDialog(&inOptions,
NULL, NULL, NULL, NULL, NULL,
&theDialog);

NavDialogRun(theDialog);
theStatus = NavDialogGetReply ( theDialog, &theReply);
NavDialogDispose(theDialog);

if(!theReply.validRecord)
{
    return -1;
// Assuming the user changed his/her mind?
// No harm; no foul, but need to know
// not to try to open the file.
}

// Get a pointer to selected file
err = AEGGetNthPtr(&(theReply.selection), 1,
typeFSS, &theKeyword,
&actualType, theFileSpec,
sizeof(FSSpec),
&actualSize);

return err;
}

```

If you want to filter files to limit the options provided the user in the Navigation Dialog, you can either use an 'open' resource created with a resource editor or you can create a NavTypeList. To use an 'open' resource you get the NavTypeListHandle with a GetResource function.

```

NavTypeListHandle typeList =
(NavTypeListHandle)GetResource('open', 128);

theStatus = NavCreateGetFileDialog(&inOptions,
typeList,
NULL, NULL, NULL, NULL,
&theDialog);

```

The 128 is just the number of the resource you have created. If no such resource is found typeList will be given a NULL value and you will be doing no file filtering.

To create your own NavTypeList from scratch you need only fill in a few blanks.

```

NavTypeList    inTypeList;
NavTypeListPtr inTypeListPtr;

inTypeList.componentSignature = kNavGenericSignature;
inTypeList.osTypeCount = 1;
inTypeList.osType[0] = 'TEXT';

inTypeListPtr = &inTypeList;

theStatus = NavCreateGetFileDialog(&inOptions,
&inTypeListPtr,
NULL, NULL, NULL, NULL,
&theDialog);

```

I have used a NavTypeListPtr just to minimize the confusion when it is necessary to pass a NavTypeListHandle to NavCreateGetFileDialog. The kNavGenericSignature is a system constant which tells it not to filter files by their creator. If you wanted to choose from only Excel files you could put 'XCEL' here instead. Since the osType is an array you can list several file types for a given application signature, and you can also make NavTypeList itself an array so that the handle tells the dialog to display any number of file types from any number of specific applications. If you need to do this, though, you are on your own.

SOFTWARE LOCALIZATION MADE *easy*

POWERGLOT FEATURE HIGHLIGHTS

- LOCALIZE CLASSIC, CARBON™, COCOA® AND PALM OS™ APPS
- LEVERAGE EXISTING TRANSLATIONS
- AUTOMATE WITH APPLESCRIPT®
- IMPORT /EXPORT TRANSLATION MEMORIES

www.powerglot.com
browse, translate, click, done.

PowerGlot Software • Localization tools for Mac® OS
www.powerglot.com
info@powerglot.com

Mac OS, Carbon, Cocoa and AppleScript are trademarks of Apple Computer, Inc.
Palm OS is a trademark of Palm, Inc.

By Clark Jackson, Tacoma Power

Using NSMutableParagraphStyle

How to programmatically set tabs and more in an NSTextView

The need to become acquainted with NSMutableParagraphStyle occurred when I attempted to automate some reporting. At times "enterprise" computing stands in contrast to typical "consumer" computing because the latter is *user-centric* while the former is *report-centric*. For example, consumer computing facilitates lots of user control whereas enterprise computing just wants to-generate-reports-unattended-at-midnight-thank-you. Since much of the text system in Cocoa "just works" for the *user*, the details of how to let the *computer* take control in making a report are often left a mystery. Let's shed a little light on some of that mystery.

What we need is to know how to use just one piece of Cocoa's extensive and capable text system that will allow a *program* to format text inside an NSTextView. The key is using the NSMutableParagraphStyle class.

SOME PRELIMINARIES

Explaining Interface Builder and Project Builder is beyond the scope of this article. Suffice it to say, the project uses the Model-View-Controller paradigm employing MyModel, MyWindowController, and NSTextView classes. Here's what the view looks like:

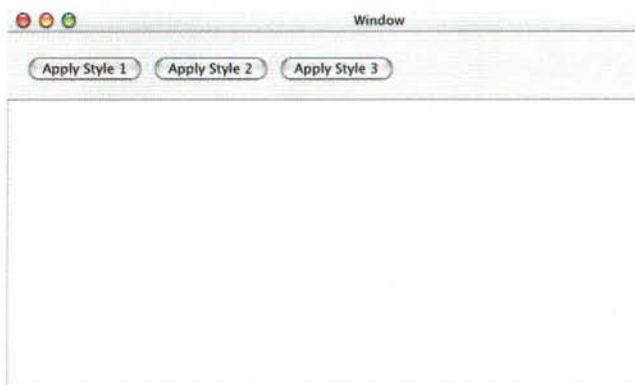


Figure 1. The interface window containing an NSTextView.

The only thing noteworthy about this window is that it contains an NSTextView because that is where our formatted text will appear.

To see the results of our formatting it will be useful to use NSTextView's ruler. To see the ruler it is necessary to add the Format menu to our menu bar. Therefore, before leaving IB, we drag the Format menu onto our project's main menu bar and give Show Ruler a key equivalent:

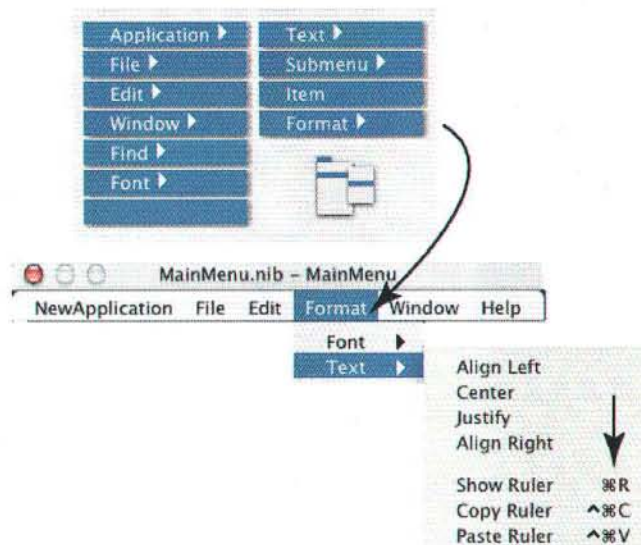


Figure 2. For convenience, in Interface Builder we drag a Format menu onto the menu bar and give Show Ruler a key equivalent.

THE VIEW

For our purposes, the important thing to know about NSTextView is that the text displayed in an NSTextView is its textStorage and that textStorage holds **attributed** strings. The textStorage is an instance of NSTextStorage which inherits from NSAttributedString itself. Attributed strings are Cocoa Foundation objects that contain both ordinary NSString's and formatting objects that apply to those strings, for example,

Clark Jackson test drove the first Mac and recognized that it was about to change the world. Lately he has been busy wearing many hats automating energy accounting in the rapidly changing electrical power industry at Tacoma Power. He can be contacted at cjackson@cityoftacoma.org.

Free On-site Setup Help!

Buy AccountEdge and get a free hour of consultation

**we'll come to your office!*

Hurry!
Limited time
offer. Visit
web for
details



MYOB AccountEdge
Evolving with the Mac since 1989
Small Business Management and Accounting

www.myob.com/us



800-322-MYOB (6962)

Small Business. Smart Solutions.®

© MYOB 2003

Void where prohibited. Cannot be combined with any other offer. Must adhere to claim form requirements and rules. *If an MYOB Certified Consultant is not available in your area, a phone consultation may be substituted.

NSFont and NSParagraphStyle. You can bypass dealing with NSTextView's textStorage directly, that is, you can [myNSTextView setString:@"aString"], but @"aString" will adopt either the default paragraph style or a previous style already residing in the textStorage.

Therefore, in order for text to appear formatted in an NSTextView we have to: 1) create an NSAttributedString instance initialized with the string we want to format, and 2) give that NSAttributedString instance the objects that it will use to format the string—in this case an NSParagraphStyle object. This attributed string goes into our NSTextView's textStorage where it appears formatted to the user.

THE CONTROLLER

The interface buttons are connected to the controller's actions in a typical fashion. Basically, each button initiates a controller action to display a different formatted paragraph in the NSTextView. The controller does this task by sending a string (NSString) to the model object. The model object will return attributed strings (NSAttributedString's attributed with NSParagraphStyle's) to the controller. Finally, the controller will pass the attributed string to the text storage of the NSTextView where it will appear formatted to the user.

Here is one of the three controller actions:

```

                                applyStyle1
This method sets a pattern for the other two belonging to the controller. An
unformatted string is put in the NSTextView for comparison purposes only. The
program pauses so the user can see it. Next, the string is sent to the model object
which returns it as an attributed string. Finally, the attributed string goes into the
NSTextView revealing the new format. Refer to the source to see the other two action
methods, applyStyle2 and applyStyle3.

- (IBAction)applyStyle1:(id)sender
{
    MyModel *theModel;
    NSAttributedString *anAttributedString;
    NSString *aString;

    // a string containing multiple paragraphs
    aString = @"On a merry-go-round in the night. ";
    aString = [aString stringByAppendingString:
        @"Coriolis was shaken with fright.\n"];
    aString = [aString stringByAppendingString:
        @"Despite how he walked. "];
    aString = [aString stringByAppendingString:
        @"'Twas like he was stalked.\n"];
    aString = [aString stringByAppendingString:
        @"By some fiend always pushing him right."];
    // limerick by David Morin, Eric Zaslow, E'beth Haley, John Golden, and Nathan Salwen,
    // http://www.aps.org/apsnews/11855.html

    // a convenience method to restore the default paragraph style to the NSTextView
    [self resetTextView];
    // install string w/o changing paragraph style for comparison
    [myNSTextView setString:aString];
    [myNSTextView display]; // force redraw
    sleep(1); // pause

    // now convert aString to an NSAttributedString by applying an NSParagraphStyle
    // using the model object
    theModel = [[MyModel alloc] init];
    anAttributedString = [theModel attributeString1:aString];
    [theModel release];

    // now that we have the attributed string, put it into the NSTextView.
    // In order to get multiple attributed strings into a textStorage use
    // [myNSTextView insertText:attributedString] instead
    [[myNSTextView textStorage]

```

```

        setAttributedString:anAttributedString];
    [myNSTextView setNeedsDisplay:YES];
    return;
}

```

THE MODEL

MyModel has a method for each of the controller's actions. It is here that we make use of NSParagraphStyle. Each method takes an ordinary NSString and returns it as an NSAttributedString so it can end up formatted in the NSTextView. When running the program, click inside the NSTextView and make the ruler visible. Some of the paragraph style attributes that we set show up on the NSTextView's ruler:

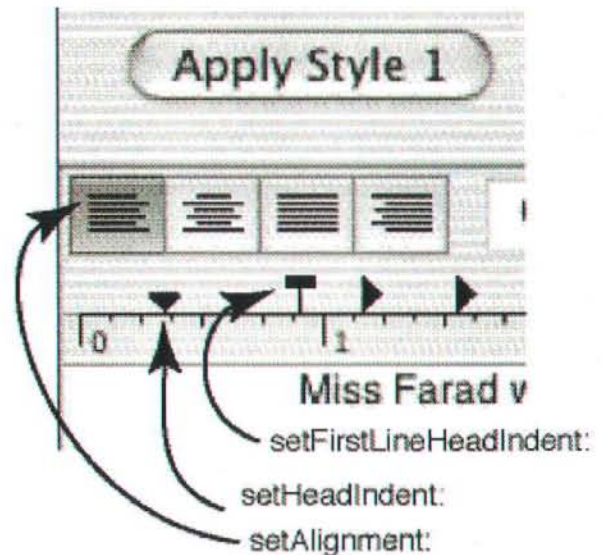


Figure 3. Many of NSParagraphStyle attributes appear in the NSTextView ruler.

Following are the methods used to create NSParagraphStyle's and apply them to strings thus creating NSAttributedString's:

```

                                attributeString1
This method creates an NSMutableAttributedString, using an NSString and an
NSMutableParagraphStyle.

- (NSMutableAttributedString *) attributeString1:
    (NSString *) aString
{
    NSMutableParagraphStyle *aMutableParagraphStyle;
    NSMutableAttributedString *attString;

    /*
    The only way to instantiate an NSMutableParagraphStyle is to mutably copy an
    NSParagraphStyle. And since we don't have an existing NSParagraphStyle available
    to copy, we use the default one.

    The default values supplied by the default NSParagraphStyle are:
    Alignment    NSNaturalTextAlignment
    Tab stops    12 left-aligned tabs, spaced by 28.0 points
    Line break mode    NSLineBreakByWordWrapping
    All others    0.0
    */

    aMutableParagraphStyle =
        [[NSParagraphStyle defaultParagraphStyle] mutableCopy];

```

Want to connect your development teams seamlessly?
You've got two choices:
Use Perforce **or** Spin your wheels



Perforce is the Fast Software Configuration Management System that gets distributed teams working together in no time.

Other systems require an army of consultants, months of hacking, and probably the odd dab of crazy glue. With Perforce, nationwide and worldwide development teams can focus on working together without worrying about holding it together.

Work anywhere. Requiring only TCP/IP, Perforce makes use of a well-tuned streaming message protocol for synchronizing client workspaces with server repository contents. And the Perforce Proxy offers caching to provide remote users with quick download times while maintaining real-time access to project activity and status information.

With Perforce, the fast way is the right way. Install it fast, learn it fast, run it fast. With other SCM systems, developers face an unpleasant choice: do it the right way or do it the fast way. Perforce's speed and reach mean the fast way is the right way. See how Perforce compares with other leading SCM systems at: <http://www.perforce.com/perforce/reviews.html>

Run at full speed even with hundreds of users and millions of files. At the core of Perforce lies a relational database with well-keyed tables, so normal user activity can be accomplished in near-zero time. Larger operations (like labeling a release and branching) are translated into keyed data access, giving Perforce the scalability that big projects require.

Develop and maintain multiple codelines. Perforce's Inter-File Branching lets you merge new features and apply fixes across codelines. Smart metadata keeps track of your evolving projects even while they are developed in parallel.

Truly cross platform. Perforce runs on more than 50 operating systems, including Windows®, UNIX®, Linux®, Mac OS® X, AS/400, and more.

Integrate with leading IDEs and defect trackers: Visual Studio.NET®, Visual C++®, Visual Basic®, JBuilder®, CodeWarrior®, TeamTrack®, Bugzilla™, ControlCenter®, DevTrack® packages, plus more.

PERFORCE
SOFTWARE

Fast Software Configuration Management www.perforce.com

Download your free 2-user, non-expiring, full-featured copy now from www.perforce.com
Free (and friendly) technical support is on hand to answer evaluation questions.

All trademarks used herein are either the trademarks or registered trademarks of their respective owners.

```
// Now adjust our NSMutableParagraphStyle formatting to be whatever we want.
//The numeric values below are in points (72 points per inch)
[NSMutableParagraphStyle
 setAlignment:NSLeftTextAlignment];
[NSMutableParagraphStyle setLineSpacing:5.5];
[NSMutableParagraphStyle setParagraphSpacing:25.5];
[NSMutableParagraphStyle setHeadIndent:25.0];
[NSMutableParagraphStyle setTailIndent:-45.0];
// setTailIndent: if negative, offset from right margin (right margin mark does
// NOT appear); if positive, offset from left margin (margin mark DOES appear)

[NSMutableParagraphStyle setFirstLineHeadIndent:65.0];
[NSMutableParagraphStyle
 setLineBreakMode:NSLineBreakByWordWrapping];

/*
possible alignments
    NSLeftTextAlignment
    NSRightTextAlignment
    NSCenterTextAlignment
    NSJustifiedTextAlignment
    NSNaturalTextAlignment

possible line wraps
    NSLineBreakByWordWrapping
    NSLineBreakByCharWrapping
    NSLineBreakByClipping
*/

// Instantiate the NSMutableAttributedString with the argument string
attString = [[NSMutableAttributedString alloc]
 initWithString:aString];

// Apply your paragraph style attribute over the entire string
[attString addAttribute:NSParagraphStyleAttributeName
 value:aMutableParagraphStyle
 range:NSMakeRange(0,[aString length])];

[aMutableParagraphStyle release]; // since it was copy'd
[attString autorelease]; // since it was alloc'd
return attString;
}
```

If your NSTextView already has attributed strings in its textStorage, you can get the NSParagraphStyle by:

```
aMutableParagraphStyle = [[myTextView typingAttributes]
 objectForKey:@"NSParagraphStyle"];
```

The NSParagraphStyle returned above comes from the attributes of the text from where the cursor is found inside the NSTextView. NSParagraphStyle's can span multiple paragraphs but there cannot be more than one per paragraph.

This method creates an array of NSTextTab's and applies them to the NSMutableParagraphStyle before it is used to create the NSMutableAttributedString.

```
-(NSMutableAttributedString *)
attributeString2:(NSString *) aString
{
    float firstColumnInch = 1.75;
    otherColumnInch = 0.6, pntPerInch = 72.0;
    int i;
    NSTextTab *aTab;
    NSMutableArray *myArrayOfTabs;
    NSMutableParagraphStyle *aMutableParagraphStyle;
    NSMutableAttributedString *attString;

    myArrayOfTabs = [NSMutableArray arrayWithCapacity:14];
    aTab = [[NSTextTab alloc]
 initWithType:NSLeftTabStopType
 location:firstColumnInch*pntPerInch];
    [myArrayOfTabs addObject:aTab];
}
```

```
[aTab release]; // aTab was alloc'd and the array owns it now so release it
for(i=1;i<14;i++)
{
    aTab = [[NSTextTab alloc]
 initWithType:NSRightTabStopType
 location:(firstColumnInch*pntPerInch)
 + ((float)i * otherColumnInch * pntPerInch)];
    [myArrayOfTabs addObject:aTab];
[aTab release]; // aTab was alloc'd and the array owns it now so release it
}

/*
possible tab stop types
    NSLeftTabStopType
    NSRightTabStopType
    NSCenterTabStopType
    NSDecimalTabStopType
*/

aMutableParagraphStyle =
[[NSParagraphStyle defaultParagraphStyle]mutableCopy];
[aMutableParagraphStyle setTabStops:myArrayOfTabs];
attString = [[NSMutableAttributedString alloc]
 initWithString:aString];
[attString addAttribute:NSParagraphStyleAttributeName
 value:aMutableParagraphStyle
 range:NSMakeRange(0,[aString length])];

[aMutableParagraphStyle release];
[attString autorelease];
return attString;
}
```

Tabbed text appears as below without using NSParagraphStyle:

	Apply Style 1	Apply Style 2	Apply Style 3	
2003/07/01 00:00:00	49	0	40	0 9 1 - 328
378 98 57.9 64	49	0	40	0 9 1 - 312
2003/07/01 01:00:00	49	0	40	0 9 4 - 301
362 89 56.9	49	0	40	0 9 1 - 294
2003/07/01 02:00:00	48	0	39	0 9 1 - 299
354 85 55.3	48	0	39	0 9 0 - 319
2003/07/01 03:00:00	48	0	39	0 9 2 - 368
343 85 54.8	48	0	39	0 9 1 - 404
2003/07/01 04:00:00	45	0	37	0 8 1 - 431
349 91 54.4	43	0	35	0 8 2 - 446
2003/07/01 05:00:00				
367 100 53.5				
2003/07/01 06:00:00				
418 112 52.7				
2003/07/01 07:00:00				
453 122 55.1				
2003/07/01 08:00:00				
477 131 58.5				
2003/07/01 09:00:00				
491 134 63.6				

Figure 4. Default behavior of an NSTextView containing tabbed text. The tab spacing is arbitrary and the lines wrap even though there is space enough to hold the whole line.



MacTech[®]
M A G A Z I N E

Get MacTech delivered to your door at a price **FAR BELOW**
the newsstand price. And, it's **RISK FREE!**

Subscribe Today!
www.mactech.com

Tabbed text appears as below after creating an NSMutableParagraphStyle and assigning it to an NSMutableAttributedString using the attributeString2 method:

Date	Time	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8	Col 9	Col 10	Col 11	Col 12
2003/07/01	00:00:00	49	0	40	0	9	1	-	328	378	98	57.9	64
2003/07/01	01:00:00	49	0	40	0	9	1	-	312	362	89	56.9	
2003/07/01	02:00:00	49	0	40	0	9	4	-	301	354	85	55.3	
2003/07/01	03:00:00	48	0	39	0	9	1	-	294	343	85	54.8	
2003/07/01	04:00:00	49	0	40	0	9	1	-	299	349	91	54.4	
2003/07/01	05:00:00	48	0	39	0	9	0	-	319	367	100	53.5	
2003/07/01	06:00:00	48	0	39	0	9	2	-	368	418	112	52.7	
2003/07/01	07:00:00	48	0	39	0	9	1	-	404	453	122	55.1	
2003/07/01	08:00:00	45	0	37	0	8	1	-	431	477	131	58.5	
2003/07/01	09:00:00	43	0	35	0	8	2	-	446	491	134	63.6	

Figure 5. By using an NSMutableParagraphStyle the tab stops have been precisely set and the lines do not wrap unexpectedly.

This method supplies a way to parse a string containing multiple paragraphs and apply different paragraph styles to each.

```
-(NSMutableArray *) attributeString3:(NSString *)aString
{
```

```
NSRange myRange,rangeOfLine,rangeOfLineContent;
unsigned startIndex,lineEndIndex,contentsEndIndex;
NSMutableParagraphStyle
    *aMutableParagraphStyle1,*aMutableParagraphStyle2;
NSMutableAttributedString *attString;
int lineCtr;
NSMutableArray * arrayOfNSMutableAttributedString;
NSString *pulledOutParagraph;
```

```
arrayOfNSMutableAttributedString
    = [NSMutableArray arrayWithCapacity:5];
```

```
/*
lines can be terminated in the following ways for use with
getLineStart:end:contentsEnd:forRange:
```

- U+000D (\r or CR), Mac
- U+2028 (Unicode line separator)
- U+000A (\n or LF), Unix
- U+2029 (Unicode paragraph separator)
- \r\n, in that order (also known as CRLF), Windows

```
*/
```

```
/*
```

```
Using getLineStart:end:contentsEnd:forRange:
```

```
@*Big\r\nMac -example string
in this case
```

```
1st contentsEndIndex is set to 3,
the first character past the line content,
line content being defined as the characters not including
line termination character(s)
```

```
1st lineEndIndex is set to 5
the first character past the end of line character(s)
```

```
*/
```

```
// make 2 different paragraph styles
aMutableParagraphStyle1 =
```

BMS

**THE LAW OFFICE OF
BRADLEY M. SNIDERMAN**

Need help safeguarding your software?

**If you're developing software, you need your valuable work protected with
trademark and copyright registration, as well as
Non Disclosure Agreements.**

**Then, when you are ready to sell it, you can protect yourself further
with a licensing agreement.**

**I am an attorney practicing in Intellectual Property, Business Formations,
Corporate, Commercial and Contract law.**

Please give me a call or an e-mail. Reasonable fees.

23679 Calabasas Rd. #558 • Calabasas, CA 91302

PHONE 818-222-0365 FAX 818-591-1038 EMAIL brad@sniderman.com

Long Distance

3.9¢ Per Minute!

Straight 6 second billing increments

Excellent rates on intrastate, intralata/toll calls and international calling with no term contract.

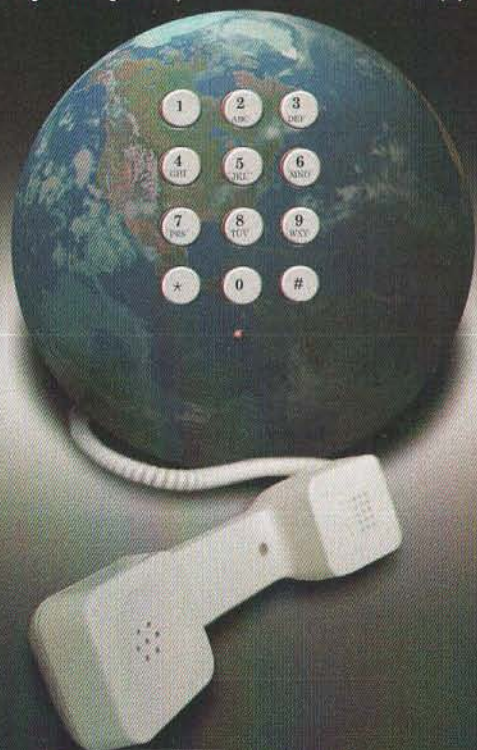
Toll Free (800/888/877/866) service, same low per minute rate for incoming calls.

10 cents per minute calling card.

Detailed billing directly from Capsule Communications, a Covista Company.

Quality electronic and telephone customer support.
No monthly billing fee if you sign up for AUTOPAY billing option or if your bill is over \$20.00 each month.

(NOTE: \$1.00 billing fee is charged when your bill is under \$20.00 for all non-Autopay customers.)



www.lowcostdialing.com

```
[ [NSParagraphStyle defaultParagraphStyle]mutableCopy];
NSMutableParagraphStyle2 =
[ [NSParagraphStyle defaultParagraphStyle]mutableCopy];
[NSMutableParagraphStyle1
 setAlignment:NSLeftTextAlignment];
[NSMutableParagraphStyle2
 setAlignment:NSRightTextAlignment];

// initialize
myRange = NSMakeRange(0,0); // (location,length)
lineEndIndex = 0;
lineCtr = 0;

/*
The while() block below walks down the string pulling out lines (paragraphs)
including their end of line (paragraph) characters, converts them to attributed
strings,
and adds them to an array.
The range in the getLineStart:contentsEnd:forRange: method has to be picky
about where it's location is but doesn't have to be picky about length (i.e. 0 will
do)
*/

while(lineEndIndex < [aString length])
{
    [aString getLineStart: &startIndex
     end: &lineEndIndex
     contentsEnd: &contentsEndIndex
     forRange: myRange];
    // rangeOfLineContent excludes line termination character(s)
    rangeOfLineContent =
        NSMakeRange(startIndex,contentsEndIndex-startIndex);
    // rangeOfLine includes line termination character(s)
    rangeOfLine =
        NSMakeRange(startIndex,lineEndIndex-startIndex);

    // apply paragraph style to pulledOutParagraph
    pulledOutParagraph =
        [aString substringWithRange:rangeOfLine];
    attString = [NSMutableAttributedString alloc]
        initWithString:pulledOutParagraph];
    if((lineCtr % 2 == 0) // alternate paragraph styles on odd/even lines
    {
        // If range.length below is zero then the paragraph style is not applied.
        // For strings with length > 0, range.length should be > 0 and <= length of the
        // string so NSMakeRange(0,1) would work as well below
        // provided you don't have an empty string.
        [attString addAttribute:NSParagraphStyleAttributeName
         value: NSMutableParagraphStyle1
         range: NSMakeRange(0,[pulledOutParagraph length])];
    }
    else
    {
        [attString addAttribute:NSParagraphStyleAttributeName
         value:NSMutableParagraphStyle2
         range:NSMakeRange(0,[pulledOutParagraph length])];
    }
    [arrayOfNSMutableAttributedString addObject:attString];
    [attString release]; // the array now owns it

    lineCtr++;
    // recalculate the range before looping
    myRange = NSMakeRange(lineEndIndex,0);
}

[NSMutableParagraphStyle1 release];
[NSMutableParagraphStyle2 release];
return arrayOfNSMutableAttributedString;
}
```

CONCLUSION

Cocoa's text system is extensive involving many classes not even mentioned here. But if you just need to get some formatted, tabbed text into an NSTextView for that special report, the methods explained here should do the trick.

***"Without a doubt, the Premiere Resource Editor
for the Mac OS ... A wealth of time-saving tools."***

– MacUser Magazine Eddy Awards

"A distinct improvement over Apple's ResEdit."

– MacTech Magazine

"Every Mac OS developer should own a copy of Resorcerer."

– Leonard Rosenthol, Aladdin Systems

***"Without Resorcerer, our localization efforts would look like a
Tower of Babel. Don't do product without it!"***

– Greg Galanos, CEO and President, Metrowerks

"Resorcerer's data template system is amazing."

– Bill Goodman, author of Smaller Installer and Compact Pro

"Resorcerer Rocks! Buy it, you will NOT regret it."

– Joe Zobkiw, author of A Fragment of Your Imagination

"Resorcerer will pay for itself many times over in saved time and effort."

– MacUser review

"The template that disassembles 'PICT's is awesome!"

– Bill Steinberg, author of Pyro! and PBTools

"Resorcerer proved indispensable in its own creation!"

– Doug McKenna, author of Resorcerer



Resorcerer® 2

Version 2.0

The Resource Editor for the Mac™ OS Wizard

ORDERING INFO

Requires System 7.0 or greater,
1.5MB RAM, CD-ROM

Standard price: \$256 (decimal)

Website price: \$128 - \$256

(Educational, quantity, or
other discounts available)

Includes: Electronic documentation
60-day Money-Back Guarantee
Domestic standard shipping

Payment: Check, PO's, or Visa/MC
Taxes: Colorado customers only

Extras (call, fax, or email us):
COD, FedEx, UPS Blue/Red,
International Shipping

MATHEMAESTHETICS, INC.
PO Box 298
Boulder, CO 80306-0298 USA
Phone: (303) 440-0707
Fax: (303) 440-0504
resorcerer@mathemaesthetics.com

**New
in
2.0:**

- Very fast, HFS browser for viewing file tree of all volumes
- Extensibility for new Resorcerer Apprentices (CFM plug-ins)
- New AppleScript Dictionary ('aete') Apprentice Editor
- MacOS 8 Appearance Manager-savvy Control Editor
- PowerPlant text traits and menu command support
- Complete AIFF sound file disassembly template
- Big-, little-, and even mixed-endian template parsing
- Auto-backup during file saves; folder attribute editing
- Ships with PowerPC native, fat, and 68K versions

- Fully supported; it's easier, faster, and more productive than ResEdit
- Safer memory-based, not disk-file-based, design and operation
- All file information and common commands in one easy-to-use window
- Compares resource files, and even **edits your data forks** as well
- Visible, accumulating, editable scrap
- Searches and opens/marks/selects resources by text content
- Makes global resource ID or type changes easily and safely
- Builds resource files from simple Rez-like scripts
- Most editors DeRez directly to the clipboard
- All graphic editors support screen-copying or partial screen-copying
- Hot-linking Value Converter for editing 32 bits in a dozen formats
- Its own 32-bit List Mgr can open and edit very large data structures
- Templates can pre- and post-process any arbitrary data structure
- Includes nearly 200 templates for common system resources
- TMPLs for Installer, MacApp, QT, Balloons, AppleEvent, GX, etc.
- Full integrated support for editing color dialogs and menus
- Try out balloons, 'ictb's, lists and popups, even create C source code
- Integrated single-window Hex/Code Editor, with patching, searching
- Editors for cursors, versions, pictures, bundles, and lots more
- Relied on by thousands of Macintosh developers around the world

To order by credit card, or to get the latest news, bug fixes, updates, and apprentices, visit our website...

www.mathemaesthetics.com

By Kenneth H. Wieschhoff, Jr.

Self-Installing Applications

Or "How I became Installer Free"

INTRODUCTION

The last step in any software development process is to create an installer. This can be Apple's PackageMaker, InstallerVise by MindVision, and the like. By making your application self-installing you can update your frameworks, libraries, (and even kernel extensions) dynamically prior to running your main application. Your users can run your application anywhere and you can include incremental updates to your software. This makes a more enjoyable user experience.

THE SECRET

In short.... Bundles! To the user your application looks like a single item, but in reality it can be an entire suite of tools. You create an Installer application that checks your support files (frameworks, kexts, libraries, et. al.) are present and up to date. Missing/outdated items are installed using Apple's Authorization Services API. Finally, this Installer launches the main application and quits. You store all the items, including the main application, in the Resources Folder within the Installer application.

THE STEPS

Here's a checklist of thing to do:

Create the self-installer application. Add your main application's icon.

Write code in main to verify your support items are installed and up to date, and then launch the main application.

Create the scripts which will install a given support item. Authorization Services will run these scripts for you at a privileged level.

Create "move" scripts to assist the build process, which move your support files and main application from their respective build directories into the Resources folder within the Installer.

CREATE THE SELF-INSTALLER APPLICATION

When I'm developing a software product that contains many different parts, I usually create a folder to hold all the

pieces. In addition to making source control easier to manage, it will simplify creating the "move" scripts. More on that later.

Here's an example of a folder layout for MyApp:

MyApplication
MyApp (the main (real) application)
MyFramework
MySelfInstaller (also produces an application called MyApp)

You'll use Project Builder to create a new "Cocoa Application" in your (new) MySelfInstaller directory. Let's have a look at the main code:

Listing 1: main.m

```
main.m
Check if support items exist and are up-to-date and then launch the main application.

#import <Cocoa/Cocoa.h>
#include <Security/Authorization.h>
#include <Security/AuthorizationTags.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/param.h>

bool      CheckItemUpToDate(char *where, char *what);
OSStatus  DoInstall(char *script);
void      LaunchMainApplication();

AuthorizationRef gAuthorizationRef = 0;

int main(int argc, const char *argv[])
{
    NSAutoreleasePool *pool=[[NSAutoreleasePool alloc] init];
    OSStatus stat = noErr;

    if ( !CheckItemUpToDate("/Library/Frameworks/",
                           "MyFramework.framework") )
        stat = DoInstall("InstallFramework.sh");

    if ( stat == noErr )
        LaunchMainApplication();

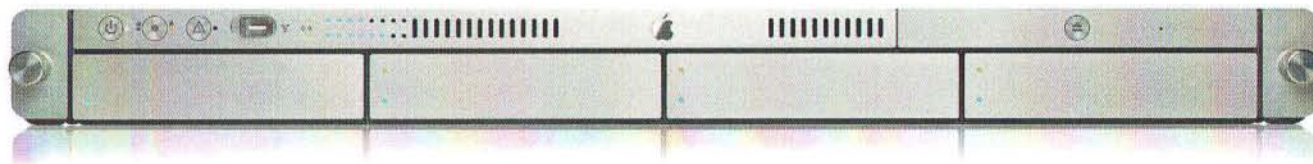
    // Release the authorization reference
    if(gAuthorizationRef)
        AuthorizationFree(gAuthorizationRef,
                        kAuthorizationFlagDefaults);

    [pool release];

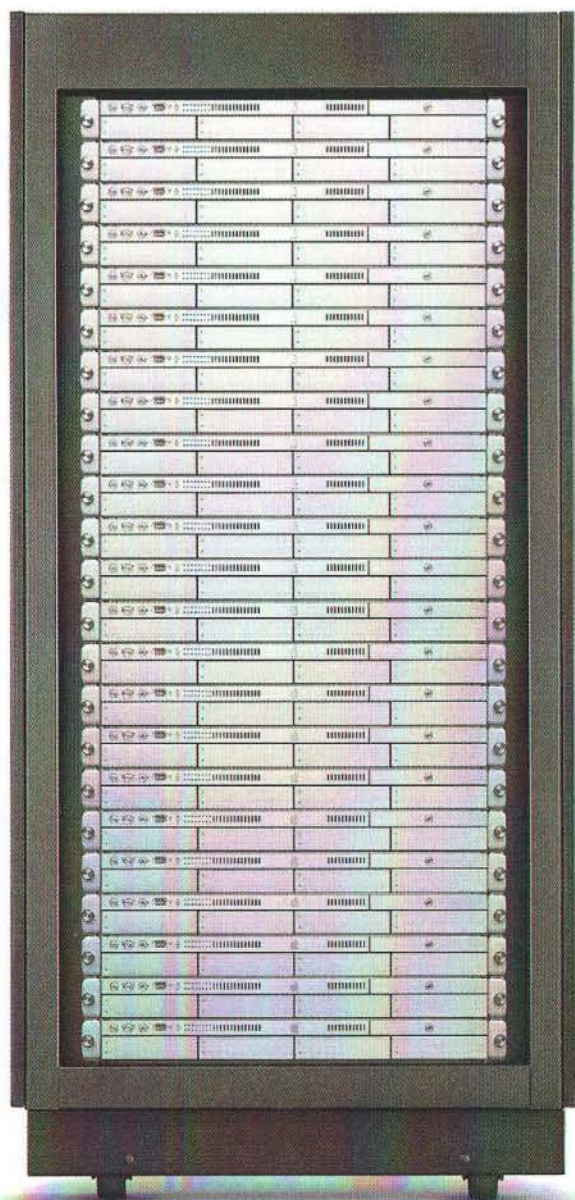
    return (EXIT_SUCCESS);
}
```

Ken Wieschhoff lives near Atlanta and works for Altea Therapeutics by day as an 8051 embedded programmer (and some Windows MFC/GUI stuff) and Eskape Labs by night where he does Cocoa programming (and occasionally device drivers) for their MyTV product line. Weekends you can find him scuba diving, riding his new Honda Valkyrie Rune, pickin' a guitar and eating grits. He can be reached at weesh@mindspring.com.

Xserve



Density optimized rack mounted Mac OS X Server



UNIX-based Server Solutions from Apple

Nearly half a terabyte of storage per 1U

630 gigaflops of processing power

Hot-Swappable drives

Industry-standard 1U rack-optimized design

There has never been a better time to buy...

Small Dog Electronics carries
factory refurbished models too
offered at substantial savings!

(subject to availability)



**Small Dog
Electronics**
www.smalldog.com

1673 MAIN STREET, ROUTE 100, WAITSFIELD, VERMONT



Apple Specialist

To learn more: <http://www.smalldog.com/xserve/>

Main checks that the framework is installed on the user's system and installs it if it's out of date or missing. It then launches the main application and then quits. (Noticeably absent in this example is some way to alert the user that installation of an item failed.)

Listing 2: main.m

main.m

The CheckItemUpToDate checks to see if a given support item is up to date.

```
bool CheckItemUpToDate(char *where, char *what){
    // Guilty until proven innocent.
    bool installNotNeeded = false;
    // Create the path to the installed support item.
    NSString *fullPath =
        [NSString stringWithFormat:@"%s%s", where, what];
    // Get the url of the "SupportItems" folder
    // within the Resources folder of our application
    CFURLRef url =
        CFBundleCopyResourceURL(CFBundleGetMainBundle(),
            CFSTR("SupportItems"), NULL, NULL);
    // Get the path to the support item within
    // our own Resources bundle
    NSString *supportPath =
        [NSString stringWithFormat:@"%s%s",
            [((NSURL *)url).path], what];
    // Get the bundle of the installed support item
    NSBundle *instExt = [NSBundle bundleWithPath:fullPath];

    // If the extension exists...
    if (instExt) {
        //...load it's dictionary.
        NSDictionary *installedDict = [instExt infoDictionary];

        // If the dictionary exists...
        if (installedDict) {
            // Get the short version string
            NSString *version = [installedDict
                objectForKey:@"CFBundleShortVersionString"];

            if (version) {
                // Do the entire thing again for the support tool
                NSBundle *local =
                    [NSBundle bundleWithPath:toolPath];

                if (local) {
                    NSDictionary *locDict = [local infoDictionary];

                    if (locDict) {
                        NSString *locVers = [locDict objectForKey:
                            @"CFBundleShortVersionString"];

                        // Compare the two strings for a match.
                        if ([locVers compare:version] == NSOrderedSame)
                            installNotNeeded = true;
                    }
                }
            }
        }
    }

    return installNotNeeded;
}
```

CheckItemUpToDate compares the Short Version Strings of the installed item against the item packaged in the bundle. Updating the version string causes the item to be re-installed. Up to date items are not re-installed.

THE MAGIC

The Authorization Services API provides a way to execute a script that needs privileges.

Listing 3: main.m

main.m

DoInstall gets authorization from the user to execute a script at a privileged level and executes the script.

```
OSStatus DoInstall(char *scriptName)
{
    char myToolPath[MAXPATHLEN];
    char *myArguments[2] = {NULL, NULL};
    FILE *myPipe = NULL;
    char myReadBuffer[128];
    AuthorizationFlags myFlags = kAuthorizationFlagDefaults;
    AuthorizationItem myItems[] = {
        // For this example we're using the standard
        // command interpreter
        {kAuthorizationRightExecute,
            strlen("/bin/sh"), "/bin/sh", 0};
    AuthorizationRights myRights = {
        sizeof(myItems)/sizeof(AuthorizationItem), myItems };
    OSStatus myStatus;

    // Tell the developer what script is being run
    printf("Running install script %s\n", scriptName);

    // Store the authorization in a global so we don't keep
    // asking the user for it once it's given.
    if ( gAuthorizationRef == 0 ) {
        // Create the authorization reference.
        myStatus = AuthorizationCreate
            ( NULL,
              kAuthorizationEmptyEnvironment,
              myFlags,
              &gAuthorizationRef);
        if (myStatus != errAuthorizationSuccess) goto bail;

        // Set flags to create our authorization environment.
        // Request to be pre-authorized to run any tool.
        myFlags = kAuthorizationFlagDefaults |
            kAuthorizationFlagInteractionAllowed |
            kAuthorizationFlagPreAuthorize |
            kAuthorizationFlagExtendRights;

        // This puts the Authorization dialog on the screen
        // and adds the authorization rights to the
        // authorization reference, gAuthorizationRef.
        myStatus = AuthorizationCopyRights
            ( gAuthorizationRef,
              &myRights,
              NULL,
              myFlags,
              NULL );
        if (myStatus != errAuthorizationSuccess) goto bail;
    }

    // If we have a valid authorization reference...
    if ( gAuthorizationRef ) {
        myFlags = kAuthorizationFlagDefaults;

        // Get the path for the script to run
        myStatus = GetScriptPath(myToolPath, scriptName);
        if (myStatus) goto bail;

        myArguments[0] = myToolPath;

        // Finally, execute our script.
        myStatus = AuthorizationExecuteWithPrivileges
            ( gAuthorizationRef,
              "/bin/sh",
              myFlags,
              myArguments,
              &myPipe);

        if (myStatus == errAuthorizationSuccess) {
            for(;;) {
                // Scripts send output back through myPipe which is
                // redisplayed on standard out
                int bytesRead = read
                    (fileno(myPipe),
                     myReadBuffer,
                     sizeof(myReadBuffer));

                // No more data!
                if (bytesRead < 1)
                    break;
            }
        }
    }
}
```

```

        break;

        write(fileno(stdout), myReadBuffer, bytesRead);
    }
    else
        printf("AuthorizationExecuteWithPrivileges "
               "returned %ld\n", myStatus);
}

bail:
    return myStatus;
}

```

DoInstall creates an Authorization reference if it doesn't already exist, and uses that authorization reference to allow the user to add privileges to execute the shell script passed to it. In addition, any output produced by the script will be echoed on standard out. This can be a valuable debugging aid when a script is giving you trouble as you can use standard shell commands like "echo" and "pwd" in your script and see the output in your Run window in Project Builder.

LAUNCH THE APPLICATION

This is accomplished by calling `[[NSWorkspace launchApplication]`.

Listing 4: main.m

main.m

LaunchMainApplication gets a path to the main application located within the bundle and launches it.

```

void LaunchMainApplication() {
    // Get the ref to the main app in the resources folder
    CFURLRef url = CFBundleCopyResourceURL(
        CFBundleGetMainBundle(),
        CFSTR("MyApp.app"),
        NULL, NULL);

    NSString *path = [NSString stringWithFormat:
        @"%@Contents/MacOS/MyApp",
        [((NSURL *)url) path]];

    [[NSWorkspace sharedWorkspace] launchApplication:path];
}

```

AN INSTALLER SCRIPT

Once the user has granted the application permission to perform privileged commands, you can do anything you need to install your support item in the script.

Warning. Your script now has omniscient powers! You can cause irreparable damage to the user's system in your script if you're not careful.

Your ONE-STOP Price Comparison for Movies!

MOVIE DEPOT sm

www.moviedepot.com

Listing 5: InstallFramework.sh

InstallFramework.sh

Here's an example script which copies a framework to the system and calls update_prebinding to calculate the locations of functions within a library so applications will launch faster.

```
#!/bin/sh
# check if the global frameworks directory exists.
# create it if not
if [ ! -d /Library/Frameworks ]; then
    # create
    mkdir /Library/Frameworks
    # set group to the administrative group
    chgrp staff /Library/Frameworks
    # allow group users to modify
    chmod 775 /Library/Frameworks
fi

# make sure the framework exists within our application
if [ -d \
    MyApp.app/Contents/Resources/SupportItems/\
    MyFramework.framework ]; then
    # check if the framework exists in /Library/Frameworks
    # delete it if it is
    if [ -d /Library/Frameworks/MyFramework.framework ]; then
        rm -rf /Library/Frameworks/MyFramework.framework
    fi

    # copy our framework to the system
    cp -Rp \
        MyApp.app/Contents/Resources/SupportItems/\
        MyFramework.framework \
        /Library/Frameworks

    # change file modes on the new framework
    chmod -R ogu+r \
        /Library/Frameworks/MyFramework.framework

    # call update_prebinding
    /usr/bin/update_prebinding -files \
        /Library/Frameworks/MyFramework.framework
else
    # Perhaps you forgot to add the file to the framework?
    echo MyFramework.framework is missing from build!
fi
```

LAST STEP — “MOVE” SCRIPTS

When you're building the Installer, Project Builder makes it easy for you to move your support items and main application into the Resources folder by adding an extra step to the build process. Select the Targets tab on the main project window and select the <MySelfInstaller> target. From the Project Menu, select “New Build Phase” and the “New Shell Script Build Phase” submenu item.

In the “Shell:” text area enter “/bin/sh” if this is the shell you normally work with. (Note you can use any shell you'd like). In the second text area enter “exec ./moveMyFramework.sh”. Let's look at an example of the script:

Listing 6: moveMyFramework.sh

MoveMyFramework.sh

This script copies the framework from a sibling directory into our SupportItems folder.

```
#!/bin/sh
# if the framework already exists, delete it
if [ -d "build/MyApp.app/Contents/Resources/\
    SupportItems/MyFramework.framework" ]; then
    rm -rf \
        "build/MyApp.app/Contents/Resources/\
        SupportItems/MyFramework.framework"
fi
```

```
# Check the “SupportItems” folder actually exists
if [ ! -d \
    "build/MyApp.app/Contents/Resources/SupportItems" ]; then
    mkdir \
        "build/MyApp.app/Contents/Resources/SupportItems"
fi

# Finally, copy the framework
cp -Rp ../MyFramework/build/MyFramework.framework \
    build/MyApp.app/Contents/Resources/SupportItems
```

Don't get confused by the multiple references to “MyApp”. Remember, the goal is to make the user unaware they're actually running multiple applications. You'll want to add your application's icons to the self-installer to complete the effect.

CONCLUSION

This approach has numerous advantages to rolling out new versions of your application and provides a very nice user experience, as the user only has to authorize once during the initial run of the application. Absent is a way to un-install the application and supporting files. This is left as an exercise for the reader.

References:

Authorization Services Reference:

http://developer.apple.com/documentation/Security/Reference/authorization_ref/

MacTech®
M A G A Z I N E

**Get MacTech delivered to your
door at a price
FAR BELOW the newsstand price.
And, it's RISK FREE!**

**Subscribe Today!
www.mactech.com**



Peachpit

Essential books for the creative community

Got a Great Idea?

Don't just think about it, do it!
We've got just the books to
help you get hands-on with the
hottest software on the market.

The Painter 8 Wow! Book

By Cher Threinen-Pendarvis

0-321-20007-1 • \$49.99 • 432 pages

Photoshop Restoration & Retouching, 2nd Edition

By Katrin Eismann & Doug Nelson

0-7357-1350-2 • \$49.99 • 384 pages

Developing Digital Short Films

By Sherri Sheridan

0-7357-1231-X • \$35.00 • 400 pages

QuarkXPress 6 Killer Tips

By Eda Warren

0-7357-1303-0 • \$29.99 • 288 pages

WebObjects 5 for Mac OS X: Visual QuickPro Guide

By Joshua Marker

0-321-11549-X • \$24.99 • 392 pages

Alert! Alert!

Mac OS X Security

By Bruce Potter, Preston
Norvell, & Brian Wotring

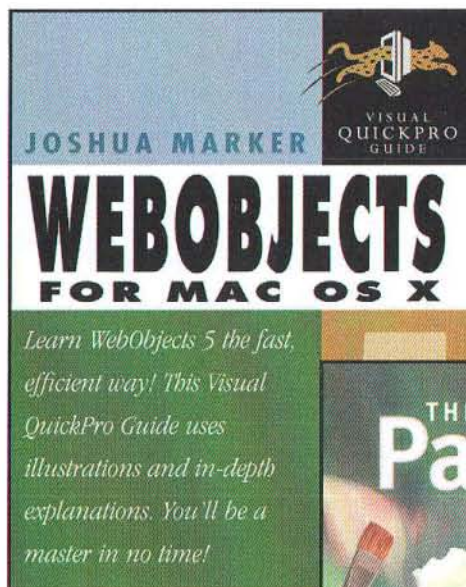
0-7357-1348-0 • \$39.99 • 408 pages



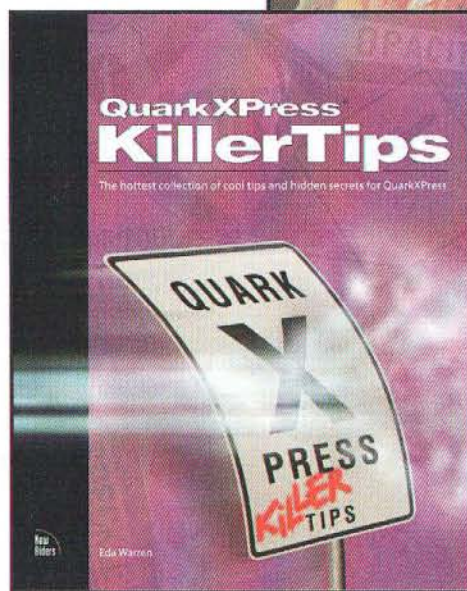
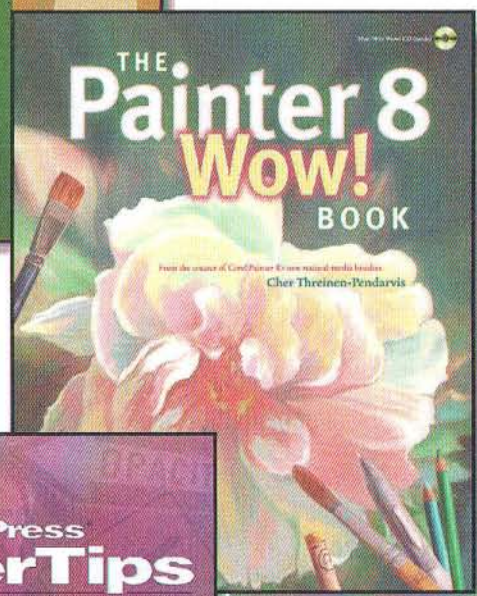
TechTV's Security Alert: Stories of Real People Protecting Themselves from Identity Theft, Viruses, and Scams

By Becky Worley

0-7357-1352-9 • \$24.99 • 288 pages



*Learn WebObjects 5 the fast,
efficient way! This Visual
QuickPro Guide uses
illustrations and in-depth
explanations. You'll be a
master in no time!*



Save up to 30%! Become a Peachpit Club Member today!

Enjoy 10% off all books every day at peachpit.com, earn an additional 10% discount as a Peachpit Club Member, and save 10% on top of that with this one-time coupon! Simply go to www.peachpit.com/mactech803 and enter coupon code EM-83AA-MTMF at checkout. It's that easy!



**New
Riders**



By Andrew S. Downs

Revolution OS

The rise of the free software and open source movements

INTRODUCTION

Every so often a movie comes along that tells a techie tale in just the right way. Making the rounds right now at film festivals and on various movie channels is an independent film called *Revolution OS*. Produced and directed by J.T.S. Moore, *Revolution OS* tells the story of the rise of the Free Software and Open Source movements. Using a combination of news bites, camera shots of Silicon Valley, and occasional statistics illustrating the increasing adoption of Linux and popularity of Linux-related Initial Public Offerings, the movie intersperses such teasers with interviews with key players in these movements.

The movie opens with Eric Raymond recounting an episode in which he encountered the Microsoft VP of Consumer Products at a conference. Eric concludes the encounter by telling the VP "I'm your worst nightmare." This opening exchange sets the tone for the movie: a growing movement that threatens the dominance of Microsoft.

The talking head format used for these interviews works better than you might expect. The main players are introduced with little fanfare. This allows you to focus on the message that each relays. We meet several of the players in the opening minutes: Eric Raymond, Linus Torvalds, Bruce Perens, and Richard Stallman. The interviews show the passion as well as the facts. The small amount of narration and voiceover provides some variety in the presentation, and in no way detracts from the message. There is no evidence of "Hollywood" in this film.

The tech talk is light. The most complex topic discussed is the distinction between the monolithic kernel architecture of Linux and the microkernel-based GNU HURD. If you read Cliff Stoll's *The Cuckoo's Egg*, the superficial treatment of potentially complex topics is similar though engaging. This makes the movie accessible to a broader audience.

For example, Bruce Perens, the author of the Open Source Definition, discusses open source as a way for developers to collaborate on projects without restrictive intellectual property laws and contracts getting in the way. Those developers sacrifice intellectual property rights in an effort to increase the

number of users of the software. This could easily have gotten bogged down in legal terminology, but Bruce makes the topic easily accessible.

We also find out that the connections between the philosopher Stallman, the engineer Torvalds, and the companies that aim to bring these products to market are based not on a bandwagon mentality but on real experience with the underlying products. Michael Tiemann of Cygnus Software and Larry Augustin of VA Linux Systems, who serve as the film's primary entrepreneurs, were both programmers who contributed to the GNU software code base earlier in their careers. No doubt this enhanced their ability to make otherwise free tools a commercial success.

Another indication of the symbiosis between companies and philosophers is Netscape's decision to release its browser source code (the Mozilla project) as open source based on the principles espoused in Eric Raymond's *The Cathedral and the Bazaar*. Fearing a Microsoft monopoly and perversion of HTML and HTTP, Netscape became the first commercial vendor to offer its source code as a product.

FREE VS. OPEN SOURCE

The movie goes to great lengths to balance the discussion between the Free Software Movement and Open Source. Although Linux and Open Source receive the lion's share of media attention, we find out that Richard Stallman and the Free Software Foundation predate Open Source by ten years or so.

Free software and open source are not synonymous. Free software is more of a political stance than an economic one: "free" refers not to price but to philosophy. Free software can be used and modified without restriction. In the proprietary, single vendor or non-free software world we know this concept as piracy, except where very liberal licenses apply. Free software began as Richard Stallman's personal crusade about twenty years ago. Interviews with Richard in the movie clearly bring out the zeal and passion in the man. What started as a configurable text editor (Emacs) in the 1980s evolved along several non-contiguous paths, including the Free Software Foundation's GNU project and the Open Source movement where Linux and Apache remain the most obvious examples of successful projects into an essential element of our technology

Andrew writes embedded system software in New Orleans, LA. You can reach him at andrew@downs.ws.

infrastructure. Some businesses base their products on code originating from free software.

Freedom to modify is a pillar of this philosophy. The responsibility accompanying that freedom comes in the form of the GNU Public License (GPL). The GPL states that distributing a product that uses free software within its core or periphery (derivatives of the GPL differ) requires the distributor or vendor to also distribute or make available the product source code so that others may enjoy the same freedom to use and modify.

MAKING MONEY

One important question with regard to both free software and open source software (which by definition is free-of-charge) is "How do you make money?" At first blush it may appear impossible. After all, I or any programmer or hacker with a compiler can download, modify, and build the finished product. Then I can simply give it to my friends and coworkers, right?

Yes, you can if the license permits it. But your next question might be "How do I configure it for my system?" Or "Why did it not install properly on my system?" If you have time and desire you may be able to determine the answers on your own. But businesses typically do not plan for, pay for, or sometimes allow this experimentation to occur on their dime. They prefer to call or email someone else for support.

And there lies the answer to the moneymaking question. The rise in outsourcing encourages firms that specialize in support (including packaging, installation, configuration, and usage issues) of open source and free software products. Businesses can then expense support costs by purchasing a packaged GNU/Linux product from a vendor such as RedHat, Caldera, SuSE, or VA Linux and receive ongoing support from outside the company. This reduces or eliminates the need for in-house expertise, though pockets of knowledge within the company will remain and likely grow.

MICROSOFT

Microsoft's role in the movie is cannon fodder for several anecdotal episodes discussing encounters between free software and open source luminaries and Microsoft managers and the company-at-large. One of the most memorable is Bill Gates' 1976 letter to the Homebrew Computer Club, in which he verbally clubs those developers who used Microsoft's BASIC compiler without paying for it. In contrast, the Windows Refund Day protest in 1999 provided less drama, since Microsoft provided drinks rather than water cannons when the protestors showed up at their building. But the movie is not a Microsoft bashfest. Rather, it illustrates the reasons behind the rise of open source and free software. These people changed the world in response to the proprietary software philosophy, of which Microsoft is the most visible proponent.

APPLE

So where is the Apple and Mac OS connection in all this? The movie discusses free software (the GNU tools) and open

source (Berkeley Unix) products that are familiar territory to Mac OS developers with the rise of OS X. The GNU development tools lie at the center of Project Builder, the development environment that ships with OS X. Berkeley Unix, itself an attempt to bring freedom to the category of operating systems by competing with AT&T's proprietary Unix, has over the years given rise to offshoots that use open source and collaboration as their development philosophy. FreeBSD is one of those offshoots and sits near the core of OS X. (The Mach microkernel resides underneath. Mach is another open source project that receives little air time in the movie but broke new ground in kernel development.)

At the other end of the Mac OS spectrum are technologies such as Aqua and QuickTime that are unlikely to become open source candidates since they are key to differentiating Mac OS from competing operating systems.

In the middle lie extensions to the kernel and BSD subsystems, including Directory Services and Rendezvous. These remain open source projects maintained by Apple and interested developers outside the company.

SUGGESTED READING

I enjoyed this movie, but it may leave some technophiles longing for more. Its treatment of technical issues is light. If you find the movie intriguing or simply want to learn more about the people, ideas, and products involved, here are several books you should consider reading:

The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary, by Eric S. Raymond, 1999, 2001. O'Reilly and Associates, Inc. The essays in this book capture the essence of the open source philosophy and attempt to quantify and explain the "why" behind the "what".
Free as in Freedom: Richard Stallman's Crusade for Free Software, by Sam Williams, 2002. O'Reilly and Associates, Inc. This book presents Richard Stallman's free software philosophy and history through recaps of interviews with the man himself and other sources.

Just for Fun: The Story of an Accidental Revolutionary, by Linus Torvalds and David Diamond, 2001. HarperCollins. Half-written by Linus Torvalds, with the balance consisting of interview recaps, this book provides insight into Linus' character and views on life, Linux, and the pursuit of family.

Open Sources: Voices from the Open Source Revolution, edited by Chris DiBona, Sam Ockman, and Mark Stone, 1999. O'Reilly and Associates, Inc. This contains essays by Stallman, Raymond, Torvalds, Perens, Tiemann and others.

Rebel Code: The Inside Story of Linux and the Open Source Revolution, by Glyn Moody, 2001. Perseus Publishing. An account of the rise of Linux from a spare-time college project to its dominance today.

Also, check out the Revolution OS web site at <http://www.revolution-os.com/>.

By Scott Knaster

Book Review: Mac OS X Hacks

When Apple decided to fuse the Mac OS with Unix and NeXT software, it created a unique stack of code that shows its varied parentage: Mac OS when you're in Aqua, Unix when you're typing at the command line in Terminal, and NeXT when you use Cocoa. But it's not just software that Apple has gathered together. Mac OS X has also brought new communities of developers and users into the Mac universe. Unix aficionados and open source advocates who never gave Apple a second thought have been drawn into the new world of OS X.

Mac OS X Hacks is one result of this new community. It's part of a series from O'Reilly, a company long known for books about cutting edge and non-proprietary (and non-Apple) technologies, and now very active in publishing Mac books since the advent of OS X. **Mac OS X Hacks** feels like a book that's truly native to its subject: not a Unix book, nor a NeXT book, and clearly not a book from OS 9 days, but made for the new Mac universe.

ARRANGEMENT

Mac OS X Hacks, subtitled **100 Industrial-Strength Tips & Tools**, is organized into nine chapters, using groupings that probably work just as well as any other: chapter titles include Files, Startup, Multimedia and the iApps, and Networking. Another useful organization might have been according to type of user: regular folks, programmers, network administrators, and so on. For example, the Networking chapter includes tips on how to share files between Macs and Windows computers, interesting to many users, and how to set up your own domain name service, which is interesting but much more specialized.

In addition to the chapter organization, the hacks are numbered consecutively from 1 to 100, and every page includes the current hack's number in a nice purple inset in the corner.

TIPS FOR THE REST OF US (AND OUR MOMS)

One of the best features of **Mac OS X Hacks** is that many of the tips are potentially useful for just about anybody running OS X. There's a tip on backing up, another on getting rid of files in the trash that don't want to leave or CDs that won't eject, and one on how to share your Internet connection. Although it's likely that most people buying a book with "Hacks" in the title are ready for advanced topics, these everyday tips are generally useful and can be passed on to the less technically savvy folks who depend on you for support.

Some chapters, in particular the one entitled The User Interface, describe how you can gain greater control of your Mac

using shareware and commercial utilities. For example, you'll find descriptions of a couple of alternatives to the Dock, and various tips on how to restore lost features from OS 9.

DEVELOPERS ONLY (UNIX FLAVORED)

Mac OS X Hacks has a fine collection of tips that are practical, but primarily useful to developers or other advanced geeks. A useful section sheds light on what's in an application package and how you can mess around with it. There are also sections on secure tunneling, remote login with SSH, and messing around with WebDAV and FTP servers.

The book does a good job of avoiding becoming a complete Unix-fest, but there are a lot of useful Unix-based hacks, including an entire chapter called Unix and the Terminal. The tip called Top 10 Mac OS X Tips for Unix Geeks is self-explanatory and useful for the incoming Unix crowd. Other sections in the Unix chapter provide information on how to get the most out of Terminal, how to use `sudo`, and how to open files and applications using the command line. This chapter is particularly useful to old Mac folks just getting a handle on Unix.

IT'S EDUCATIONAL

Some of the hacks presented in **Mac OS X Hacks** are not practical tips per se, but instead provide overview information on how particular features work. Although these tips might not be immediately useful, they're quite handy for increasing your general OS X knowledge. The very first hack, Understanding and Hacking Your User Account, gives a good overview of this useful topic which is rather alien to OS 9 folks. Another section updates file type and creator codes with OS X information. A couple of sections provide nifty insight into what goes on when OS X starts up.

IS THIS TIP REALLY NECESSARY?

With 100 hacks, not every one will be useful, educational, or fun. It's a safe bet that not a lot of readers will really be interested in installing the PostgreSQL database or Setting a Password in Open Firmware. Still, even these topics can satisfy your intellectual curiosity.

CONCLUSION

The test of most "tips & tricks" books is in their practicality. The contents should be fun and interesting, but if they're useful too, the book is a success. **Mac OS X Hacks** does an excellent job of delivering a solid package of topics that are interesting *and* useful.



Keep track of every second of your time and bill for it with Time Track! Built in instructions make it easy to use. It is a simple way to manage your billable time for multiple projects and create a web page to show to your clients. Only \$24.95 per single user license per platform. Finally! A versatile time tracking solution for Macintosh, Windows, and Palm!

www.trinfinitysoftware.com



By TLA Systems

The Dock with more than one dimension.

Create multiple docks of any size, assign hot keys and even put the Trash back on the Desktop. A flexible and feature-laden tool for power-users. Runs natively on Mac OS X and 9 in five languages.

"...you made the switch to OS X a lot easier for me..." - Bob LeVitus

"...DragThing can rightfully be called an indispensable aid to working with your Mac..." - MacUser UK

Download a copy now from www.dragthing.com.



Trapcode - plug-ins for Adobe® After Effects®

Trapcode Shine is a fast light effect plug-in. The effect looks very much like volumetric light, but is actually a 2D effect. There are special controls to make shimmering lights and numerous coloring modes. This is an effect that you see everyday on TV and in many movie titles. Shine is available for Mac, Mac OSX and Windows.

www.trapcode.com



Eudora Internet Mail Server (EIMS) 3.2 is the latest version of the most popular Internet mail server for the Macintosh. If you need to handle email for a dozen users, or thousands of users, EIMS is a reliable and easy to use solution.

EIMS 3.2 is available for US\$400.00, there are no limits on the number of users that can be added, and free email support is included.

For more information, see

<http://www.eudora.co.nz/>



GraphicConverter converts pictures to different formats. Also it contains many useful features for picture manipulation.

See www.lemkesoft.com
for more information.

List of Advertisers

Aladdin Knowledge Systems, Inc.	13
Aladdin Systems, Inc.	81
Big Nerd Ranch, Inc.	57
Brad Sniderman	67
Circus Ponies Software, Inc.	35
DevDepot	28-29
digitalforest	21
EazyDraw (Dekorrra Optics, LLC)	47
effigent, Inc.	15
Electric Butterfly	36
Eudora Internet Mail Server	79
FairCom Corporation	1
Felt Tip Software	19
Fetch Softworks	7
Full Spectrum Software, Inc.	23
James Sentman Software	42
Jiiva, Inc.	39
Lemke Software GmbH	79
Lingo Systems	41
MacDirectory	51
MacTech Magazine	59
Mathemaesthetics, Inc.	69
MCF Software	27
Movie Depot	73
MYOB US, Inc.	63
/n software inc.	43
Netopia, Inc.	IFC
Paradigma Software	31
Parliant Corporation	45
Peachpit Press	75
Perforce Software, Inc.	65
piDog Software	56
PowerGlot Software	61
Presto Vivace, Inc.	23
PrimeBase (SNAP Innovation)	55
Prosoft Engineering, Inc.	49
Redstone Software, Inc.	9
Runtime Revolution Limited	82
Seapine Software, Inc.	37
Small Dog Electronics	71
Sophisticated Circuits, Inc.	33
Sybase, Inc.	2-3
The Iconfactory	12
ThinkFree Corporation	25
TLA Systems Ltd.	79
Trapcode Software	79
Trinfinity Software	79
Trolltech AS	11
Utilities4Less.com	68
VVI	17
WIBU-SYSTEMS AG	53

List of Products

Adobe Press • Peachpit Press	75
Big Nerd Ranch • Big Nerd Ranch, Inc.	57
c-tree Plus • FairCom Corporation	1
Development & Testing • Full Spectrum Software, Inc.	23
Disk Utilities • Prosoft Engineering, Inc.	49
DragThing • TLA Systems Ltd.	79
Eazy Draw • EazyDraw (Dekorrra Optics, LLC)	47
Eggplant • Redstone Software, Inc.	9
EIMS • Eudora Internet Mail Server	79
Enterprise Software • MCF Software	27
Fetch • Fetch Softworks	7
Graphic Converter • Lemke Software GmbH	79
Hosting Solutions • digitalforest	21
InstallerMaker, StuffIt • Aladdin Systems, Inc.	81
IP*Works! • /n software inc.	43
Law Offices • Brad Sniderman	67
Long Distance Phone Service • Utilities4Less.com	68
MacDirectory • MacDirectory	51
MacTech Magazine Subscription • MacTech Magazine	59
Maximizing Your Mac! • DevDepot	28-29
moviedepot.com • Movie Depot	73
MYOB • MYOB US, Inc.	63
Notebook • Circus Ponies Software, Inc.	35
Phone Valet • Parliant Corporation	45
piDog Utilities • piDog Software	56
PowerGlot • PowerGlot Software	61
PowerKey Pro & KickOff! • Sophisticated Circuits, Inc.	33
Presto Vivace Services • Presto Vivace, Inc.	23
PrimeBase • PrimeBase (SNAP Innovation)	55
Qt • Trolltech AS	11
Resorcerer • Mathemaesthetics, Inc.	69
Revolution • Runtime Revolution Limited	82
SCM Software • Perforce Software, Inc.	65
Security • Aladdin Knowledge Systems, Inc.	13
SmallDog.com • Small Dog Electronics	71
Software Protection • WIBU-SYSTEMS AG	53
Sound Studio • Felt Tip Software	19
Stock Icons • The Iconfactory	12
SuperScrubber & Application Builder Collection • Jiiva, Inc.	39
Sybase services • effigent, Inc.	15
Sybase • Sybase, Inc.	2-3
TestTrack Pro • Seapine Software, Inc.	37
ThinkFree • ThinkFree Corporation	25
Timbuktu & netOctopus • Netopia, Inc.	IFC
Time Track • Trinfinity Software	79
Translation & Localization • Lingo Systems	41
Trapcode • Trapcode Software	79
UniHelp Module • Electric Butterfly	36
Valentina • Paradigma Software	31
Visual-Report Tool Developer • VVI	17
WhistleBlower • James Sentman Software	42

The index on this page is provided as a service to our readers. The publisher does not assume any liability for errors or omissions.

Multiple formats. Multiple platforms. Complex installers.

Aladdin solves the compression and installation puzzle.

**Trying to figure out how to handle multiple
compression formats and platforms?**

The Stuffit Engine solves the compression puzzle.



Aladdin's Stuffit Engine SDK:

- Adds value to your application by integrating powerful compression and encryption.
- Is the only tool that supports the Stuffit file format.
- Provides a single API that supports over 20 compression and encoding formats common on Macintosh, Windows, and Unix.
- Makes self-extracting archives for either Macintosh or Windows.
- Available for Macintosh, Windows, Linux, or Solaris.

Licenses start as low
as \$99/year

To learn more, visit:
www.stuffit.com/sdk/

Stuffit Engine SDK™ The power of Stuffit in your software.



**Looking for the easiest and fastest
way to build an installer?**

Stuffit InstallerMaker completes your puzzle.

It's not enough just to write solid code anymore. You still have to write an installer for your users. Stuffit InstallerMaker makes it simple and effective.

- Stuffit InstallerMaker gives you all the tools you need to install, uninstall, resource-compress or update your software in one complete, easy-to-use package.
- Add marketing muscle to your installers by customizing your electronic registration form to include surveys and special offers.
- Make demoware in minutes. Create Macintosh OS X and Macintosh Classic compatible installers with Stuffit InstallerMaker.

Prices start at \$250

To learn more, visit:
[www.stuffit.com/
installermaker/](http://www.stuffit.com/installermaker/)

Stuffit InstallerMaker™ The complete installation solution.™



www.stuffit.com
(831) 761-6200

© 2002 Aladdin Systems, Inc. Stuffit, Stuffit InstallerMaker, and Stuffit Engine SDK are trademarks of Aladdin Systems, Inc. The Aladdin logo is a registered trademark. All other products are trademarks or registered trademarks of their respective holders. All Rights Reserved.

It'll Give You A Life.

Nassau Beach, Bahamas - Sun, sand and soft breezes make this one of the most relaxing vacation spots in the world. Don't forget the cool, fruity drink!



Dear Revolution 2.0,

Thanks for the
great vacation!

Missing you terribly,

Love,

Cecil

Revolution 2.0
c/o My Computer 64.23.0.192
24-7 Relief From Aggravation
(formerly DullAndDreary Coders)
The Corporate World, #1-EASY

But A Geek Is Always A Geek.

**Revolution 2.0: the English like language designed around the way you think.
Develop and deliver on Mac OS X, Windows and Linux
(not to mention 10 other platforms).**

Take the English language, add elegant XML, more SQL databases than you care to learn, intrinsic video capture, Unicode, CGI scripting, sophisticated Reports, and build your solution faster than anyone else. Jaguar, Panther, XP? Revolution lets you eat platforms for breakfast. And speaking of eating, our Cookbook of examples gets you up, running, and productive NOW. You can join the Revolution for as little as \$99...
Build with it, deliver with it, love it - and still have time for vacations.

And for a limited time, your MacTech special lets you get 100% of the Revolution for 15% off - go to special.runrev.com NOW. Thousands of developers have already joined.
Don't let the Revolution in modern coding start without you...

Software At The Speed Of Thought



Software Development & Consultancy

Runtime Revolution • 91 Hanover Street • Edinburgh EH2 1DJ • UK
Phone +44 (0)131 718 4333 • Fax +44 (0)131 718 4334 • www.runrev.com • Email info@runrev.com